University of Alberta

NEURAL NETWORK APPROACH FOR NONLINEAR DYNAMICS PREDICTION AND FEATURE EXTRACTION

by

Ovidiu Voitcu ©

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Applied Mathematics

Department of Mathematical and Statistical Sciences
Edmonton, Alberta
Fall, 2004

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-96035-8
Our file Notre référence
ISBN: 0-612-96035-8

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

To Claudia

ACKNOWLEDGEMENT

I would like to thank my supervisor Dr. Yau Shu Wong for his support and for providing me with the opportunity to acquire mathematical modelling, computing, and practical problem solving skills, by proposing the present research topic. I also wish to thank Dr. Peter Minev, Dr. Yanping Lin, Dr. Henry van Roessel, Dr. Chongqing Ru, and Dr. Jianhong Wu for accepting to be members of my Ph.D. examination committee and for taking the time to read my thesis. Many thanks to Dr. Wong, Dr. Minev, Dr. Lin, Dr. Wu, Dr. B. H. K. Lee, and Dr. Abel Cadenillas for generously supporting me with letters of reference for scholarship and job applications.

This work is supported by the Natural Sciences and Engineering Research Council of Canada. I wish to thank Dr. Thomas W. Strganac and the Aeroelasticity Group at the Texas A & M University for making their test data available and Dr. Liping Liu for providing the computer code for numerically generating test signals. I also wish to thank Dr. B. H. K. Lee of the National Research Council of Canada for providing us with the test signals for the cavity oscillations. I would like to thank my colleagues Dr. Cristina Popescu and Dr. Liping Liu for their useful exchange of information in approaching this research topic from different technical perspectives. Many thanks to Dr. Hongtao Yang for sharing this LaTeX thesis format with his colleagues.

Finally, I could never find enough words to thank my life partner and best friend Claudia Caia for standing by me and offering me the gift of her love throughout this challenging period of my life. Her kindness and understanding gave me more strength to overcome countless obstacles.

Table of Contents

1 Introduction **1**

1.1 Prediction of Nonlinear Oscillations 1

1.2 Connectionism Versus Symbolism 3

1.3 Applications of Neural Networks 5

1.4 Original Contributions 9

2 Artificial Neural Networks (ANNs) **13**

2.1 Mathematical Representation of ANNs 13

2.2 ANN Training 16

2.3 The Universal Approximation Property of ANNs 21

3 Main Contribution **26**

3.1 Time Series Models 26

3.2 The Proposed Approach 29

3.3 Implementation for ANN Prediction 33

3.4 Dynamical Systems Perspective 42

4 Case Studies **57**

4.1 Test Cases 58

4.2 Choice of Methods and Parameters 62

4.3 Prediction Accuracy 65

4.4 Robustness Comparison 68

5 Further Applications **90**

5.1 Cavity Oscillations 90

5.2 Feature Extraction 95

6 Conclusions and Future Directions **99**

Bibliography **102**

List of Tables

4.1 ANN Architecture and Training Parameters. 59
4.2 ANN MSP Methods 62
4.3 Local Minimum and Local Maximum Points in the Test Signals. 63
4.4 Selection of Network Inputs 67
4.5 Time History -based Prediction Accuracy Scores (THPAS). 68
4.6 Phase Portrait -based Prediction Accuracy Scores (PPPAS). 69
4.7 Overall Prediction Accuracy Scores (OAPAS). 70

List of Figures

2.1 Biological Neuron. 13
2.2 Artificial Neuron. 13
2.3 Two Layer Feedforward ANN. 15

3.1 Two Layer Feedforward ANN Used for Prediction. 30
3.2 The function $\psi(x) = \ln\{1 + e^{x}/(1 + e^{-x})\}$ ('—'). 34
3.3 ANN Training Set with Lmax and Lmin Points (Test Signal S1). 39
3.4 ACF for an ANN Training Set (Test Signal S3). 40
3.5 FFT Plot of a Training Set (Test Signal S5). 41
3.6 Test Signal ('—'), and MSP ('- -'). 41
3.7 Sample $\rho^{[\cdot]}$ (a) and $\bar{\rho}_5^{[\cdot]}$ (b). 42
3.8 MSP for $n_1 = 3$ ('—'), 4('- -'), 5('-.'), 6('...'). 42
3.9 Phase plot for training set (left) and testing set (right). 50

4.1 Test Signal S1. 59
4.2 Test Signal S3. 59
4.3 Test Signal S5. 60
4.4 Training Sets: Time History and Phase Plot for S1, Time History for S2. 60
4.5 Training Sets: Time History and Phase Plot for S3, Time History for S4. 60
4.6 Training Sets: Time History and Phase Plot for S5, Time History for S6. 61
4.7 Test Signal S7. 61
4.8 Training Sets: Time History and Phase Plot for S7, Time History for S8. 61
4.9 ACF plot for S1-S4. 64
4.10 ACF plot for S5-S8. 64
4.11 FFT plot for S1 and S2. 65
4.12 FFT plot for S3 and S4. 65
4.13 FFT plot for S5 and S6. 66
4.14 FFT plot for S7 and S8. 66
4.15 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M100. 72
4.16 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M101. 72

4.17 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M110. 72

4.18 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M111. 72

4.19 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M200. 73

4.20 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M201. 73

4.21 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M210. 73

4.22 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M211. 73

4.23 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M300. 74

4.24 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M301. 74

4.25 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M310. 74

4.26 S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M311. 74

4.27 S3 ('—'), and MSP for S3 ('- -'), for M100. For S4, the ANN training did not converge. 75

4.28 S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M101. 75

4.29 S3 ('—'), and MSP for S3 ('- -'), for M110. For S4, the ANN training did not converge. 75

4.30 S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M111. 75

4.31 S3 ('—'), and MSP for S3 ('- -'), for M200. For S4, the ANN training did not converge. 76

4.32 S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M201. 76

4.33 S3 ('—'), and MSP for S3 ('- -'), for M210. For S4, the ANN training did not converge. 76

4.34 S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M211. 76

4.35 S3 ('—'), and MSP for S3 ('- -'), for M300. For S4, the ANN training did not converge. 77

4.36 S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M301. 77

4.37 S3 ('—'), and MSP for S3 ('- -'), for M310. For S4, the ANN training did not converge. 77

4.38 S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M311. 77

4.39 S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M101. 78

4.40 S5 ('—'), and MSP for S5 ('- -'), for M110. For S6, the ANN training did not converge. 78

4.41 S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M111. 78

4.42 S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M201. 78

4.43 S5 ('—'), and MSP for S5 ('- -'), for M210. For S6, the ANN training did not converge. 79

4.44 S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M211. 79

4.45 S5 ('—'), and MSP for S6 ('-.'), for M301. For S5, the ANN training did not converge. 79

4.46 S5 (’—’), MSP for S5 (’- -’), and MSP for S6 (’-.’), for M311. 79
4.47 S7 (’—’), and MSP for S7 (’- -’), for M100. For S8, the ANN training did not converge. 80
4.48 S7 (’—’), MSP for S7 (’- -’), and MSP for S8 (’-.’), for M101. 80
4.49 S7 (’—’), and MSP for S7 (’- -’), for M110. For S8, the ANN training did not converge. 80
4.50 S7 (’—’), and MSP for S7 (’- -’), for M111. For S8, the ANN training did not converge. 80
4.51 S7 (’—’), and MSP for S7 (’- -’), for M200. For S8, the ANN training did not converge. 81
4.52 S7 (’—’), MSP for S7 (’- -’), and MSP for S8 (’-.’), for M201. 81
4.53 S7 (’—’), and MSP for S7 (’- -’), for M210. For S8, the ANN training did not converge. 81
4.54 S7 (’—’), and MSP for S7 (’- -’), for M211. For S8, the ANN training did not converge. 81
4.55 S7 (’—’), and MSP for S7 (’- -’), for M300. For S8, the ANN training did not converge. 82
4.56 S7 (’—’), MSP for S7 (’- -’), and MSP for S8 (’-.’), for M301. 82
4.57 S7 (’—’), and MSP for S7 (’- -’), for M310. For S8, the ANN training did not converge. 82
4.58 S7 (’—’), and MSP for S7 (’- -’), for M311. For S8, the ANN training did not converge. 82
4.59 MSP of S1 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$(’...’), 0.9(’- -.’), 1.0(’–’), 1.1(’-.’), 1.2(’- -’). . 83
4.60 MSP of S2 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$(’...’), 0.9(’- -.’), 1.0(’–’), 1.1(’-.’), 1.2(’- -’). For M101 with $\delta = 1.1$, the ANN training did not converge. 83
4.61 MSP of S3 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$(’...’), 0.9(’- -.’), 1.0(’–’), 1.1(’-.’), 1.2(’- -’). . 83
4.62 MSP of S4 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$(’...’), 0.9(’- -.’), 1.0(’–’), 1.1(’-.’), 1.2(’- -’). For M101 with $\delta = 1.1$ and 1.2, the ANN training did not converge. 83
4.63 MSP of S5 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$(’...’), 1.0(’–’), 1.1(’-.’), 1.2(’- -’). For M101 with $\delta = 0.9$ and for M201 with $\delta = 0.8$ and 0.9, the ANN training did not converge. 84
4.64 MSP of S6 for $n_0 = \delta \times n_0^*$, $\delta = 0.9$(’- -.’), 1.0(’–’), 1.1(’-.’) only. 84
4.65 MSP of S6 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$(’...’), 1.0(’–’), 1.2(’- -’) only. 84
4.66 MSP of S7 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$(’...’), 0.9(’- -.’), 1.0(’–’), 1.1(’-.’), 1.2(’- -’). . 84
4.67 MSP of S8 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$(’...’), 0.9(’- -.’), 1.0(’–’), 1.1(’-.’), 1.2(’- -’). . 85
4.68 MSP of S1 for $n_1 = n_1^* + \delta$, $\delta = 0$(’—’), 1(’- -’), 2(’-.’), 3(’...’). 85
4.69 MSP of S2 for $n_1 = n_1^* + \delta$, $\delta = 0$(’—’), 1(’- -’), 2(’-.’), 3(’...’). 85
4.70 MSP of S3 for $n_1 = n_1^* + \delta$, $\delta = 0$(’—’), 1(’- -’), 2(’-.’), 3(’...’). 85
4.71 MSP of S4 for $n_1 = n_1^* + \delta$, $\delta = 0$(’—’), 1(’- -’), 2(’-.’), 3(’...’). 86

4.72 MSP of S5 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...'). 86
4.73 MSP of S6 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...'). 86
4.74 MSP of S7 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...'). 86
4.75 MSP of S8 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...'). 87
4.76 MSP of S1 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -'). 87
4.77 MSP of S2 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -'). For M101 with $\delta = -0.1$, the ANN training did not converge. 87
4.78 MSP of S2 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.1$('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -') only. For M101 with $\delta = -0.1$, the ANN training did not converge. 87
4.79 MSP of S3 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -'). 88
4.80 MSP of S4 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.2('- -'). For M101 and M201 with $\delta = +0.1$, the ANN training did not converge. 88
4.81 MSP of S5 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.1$('- -.'), 0.0('–'), 0.1('-.') only. 88
4.82 MSP of S5 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), 0.0('–'), 0.2('- -') only. 88
4.83 MSP of S6 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -'). 89
4.84 MSP of S7 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -'). 89
4.85 MSP of S8 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -'). 89

5.1 First 2000 data points of S9 and S10. 91
5.2 FFT for S9 and S10, based on the first 2000 data points. 92
5.3 FFT (truncated plot) for S9 and S10, based on the first 2000 data points. 92
5.4 ACF for S9 and S10, based on the first 2000 data points. 93
5.5 FFT (sampling step = 1) for S9 and S10, based on the first 2000 data points. . 93
5.6 Correct signal ('—') and MSP ('- -') for S9 and S10. 94
5.7 FFT for S9 and the ANN-MSP, based on the observations at the moments $t_1 + 1, \ldots, t_1 + n_0$. 94
5.8 FFT for S9 and the ANN-MSP, based on the observations at the moments $t_1 + n_0 + 1, \ldots, t_1 + 2n_0$. 94
5.9 FFT for S10 and the ANN-MSP, based on the observations at the moments $t_1 + 1, \ldots, t_1 + 2n_0$. 95
5.10 Simulated two-mode signal ('—') and ANN prediction ('- -'). 96
5.11 A Wavelet-ANN model for feature extraction. 96

5.12 Training ((a),(b)) and testing ((c),(d)) relative errors when using 513 ((a),(c)) and 9 ((b),(d)) network inputs respectively, in the damping extraction problem. . . . 98

Nomenclature

$\tilde{e}_{\mathbf{w}}(t)$	current OSP error $s(t) - \tilde{y}_{\mathbf{w}}(t)$
$\hat{e}_{\mathbf{w}}(t)$	current MSP error $s(t) - \hat{y}_{\mathbf{w}}(t)$
$\dot{f}$	the derivative function of $f : \mathbf{R} \to \mathbf{R}$
$f^{[m]}$	the m-fold composition $f \circ f \circ \ldots \circ f$
$f^{[0]}$	the identity function, by definition
$f^{(1)}$, $f^{(2)}$	TF of the 1st and 2nd layer, respectively
n_0	number of network inputs
n_1	number of neurons in the 1st layer of an ANN
$s(t)$	current observation of the given time series
$s(t_0+1)$, $s(t_1)$	1st and last training value, respectively
$s(t_1+1)$, $s(t_2)$	1st and last value to be predicted, respectively
t	current time step
$w^{(1)}_{k,h}$, $w^{(2)}_{k}$	weights of the 1st and 2nd layer, respectively
$\tilde{y}_{\mathbf{w}}(t)$	current OSP, $\Phi_{\mathbf{w}}\left(s(t-1), \ldots, s(t-n_0)\right)$
$\hat{y}_{\mathbf{w}}(t)$	current MSP, $\Phi_{\mathbf{w}}\left(\hat{y}_{\mathbf{w}}(t-1), \ldots, \hat{y}_{\mathbf{w}}(t-n_0)\right)$
$y^{(1)}_{\mathbf{w},k}(t)$, $y^{(2)}_{\mathbf{w}}(t)$	current output of the 1st and 2nd layer, respectively
$E(\mathbf{w})$	mean squared OSP error (performance index)
$\Phi_{\mathbf{w}}(\mathbf{x})$	the quantity $\Phi_{\mathbf{w}}(x_1, \ldots, x_{n_0})$
$\tilde{\mathbf{e}}_{\mathbf{w}}(t)$	the vector $[\tilde{e}_{\mathbf{w}}(t), \tilde{e}_{\mathbf{w}}(t-1), \ldots, \tilde{e}_{\mathbf{w}}(t-n_0+1)]^T$
$\hat{\mathbf{e}}_{\mathbf{w}}(t)$	the vector $[\hat{e}_{\mathbf{w}}(t), \hat{e}_{\mathbf{w}}(t-1), \ldots, \hat{e}_{\mathbf{w}}(t-n_0+1)]^T$
$\mathbf{s}(t)$	the vector $[s(t), s(t-1), \ldots, s(t-n_0+1)]^T$
$\mathbf{s}^{++}(t)$	the vector $[s(t), s(t-1), \ldots, s(t-n_0)]^T$
$\mathbf{w}$	the vector of all weights of an ANN
$\mathbf{w}^{(1)}_{k_1,\cdot}$	the vector $\left[w^{(1)}_{k_1,1}, w^{(1)}_{k_1,2}, \ldots, w^{(1)}_{k_1,k_0}\right]^T$
$\mathbf{x}$	generic vector $[x_1, \ldots, x_{n_0}]^T \in \mathbf{R}^{n_0}$
$\mathbf{y}(t)$	the vector $[y(t), y(t-1), \ldots, y(t-n_0+1)]^T$
$\tilde{\mathbf{y}}_{\mathbf{w}}(t)$	the vector $[\tilde{y}_{\mathbf{w}}(t), \tilde{y}_{\mathbf{w}}(t-1), \ldots, \tilde{y}_{\mathbf{w}}(t-n_0+1)]^T$

$\hat{\mathbf{y}}_{\mathbf{w}}(t)$	the vector $[\hat{y}_{\mathbf{w}}(t), \hat{y}_{\mathbf{w}}(t-1), \ldots, \hat{y}_{\mathbf{w}}(t-n_0+1)]^T$
$\hat{\mathbf{y}}(\mathbf{w})$	the vector $[\hat{y}_{\mathbf{w}}(t_1+1), \hat{y}_{\mathbf{w}}(t_1+2), \ldots, \hat{y}_{\mathbf{w}}(t_2)]^{\mathrm{T}}$
$\mathbf{N}_0$	the set of positive integers
$\mathbf{R}$	the set of real numbers
$\mathbf{\Phi}_{\mathbf{w}}(\mathbf{x})$	the vector $[\Phi_{\mathbf{w}}(\mathbf{x}), x_1, \ldots, x_{n_0-1}]^T$
^{T}	the transpose operator
$\vert\vert\vert f \vert\vert\vert_C$	the sup-norm, $\sup\{\vert f(\mathbf{x})\vert; \mathbf{x} \in \mathbf{R}^{n_0}\}$, of f
$\Vert\mathbf{x}\Vert_2$	the Euclidean norm, $\{\sum_{h=1}^{n_0} \vert x_h\vert^2\}^{1/2}$, of $\mathbf{x}$
$\Vert\mathbf{x}\Vert_\infty$	the ∞-norm, $\max\{\vert x_h\vert; 1 \leq h \leq n_0\}$, of $\mathbf{x}$
ACF	autocorrelation function
ANN	artificial neural network
ALR	adaptive LR
CLR	constant LR
FFT	Fast Fourier Transform
Lmax	local maximum
Lmin	local minimum
LR	learning rate
LRGFNN	locally recurrent globally feedforward ANN
LTMSP	long-term MSP
MLFFNN	multi-layer feed-forward ANN
MSAP	multi-step-ahead prediction
MSP	multi-step prediction
OSP	one-step prediction
OAPAS	overall PAS (average of THPAS and PPPAS)
PAS	prediction accuracy score
PP	phase portrait
PPPAS	PP-based PAS
TF	transfer function
TH	time history
THPAS	TH-based PAS
2LFFNN	two-layer feed-forward ANN
2LFF1SNN	2LFFNN with sigmoidal TF in the 1st layer
2LFF1S2LNN	2LFF1SNN with linear TF in the 2nd layer
2LFF1S2SNN	2LFF1SNN with sigmoidal TF in the 2nd layer

Chapter 1

Introduction

1.1 Prediction of Nonlinear Oscillations

A wide variety of physical and economical phenomena can be modeled by nonlinear dynamical systems[1]. Understanding nonlinear dynamics is important for predicting and controlling these complex phenomena. A particularly interesting feature associated with nonlinear dynamics is the occurrence of nonlinear oscillations. A stable oscillation may become a limit cycle oscillation due to the change in system parameters. Furthermore, a limit cycle oscillation may lead to an unstable oscillation in a later stage, and this may cause instability in the corresponding physical system.

The study of nonlinear oscillations is important in nonlinear aeroelasticity [140], which is a research field with great impact on aircraft safety. Aeroelasticity studies the mutual interaction among inertial, elastic and aerodynamic forces. Aeroelastic phenomena occur in physical systems such as suspension bridges, aircraft wings, etc. The complex aeroelastic response is governed by a nonlinear dynamical system. The nonlinearities in the system can be due to the structural or aerodynamic forces. One of the most important aspects in nonlinear aeroelasticity is to detect limit cycle oscillations. These are sustained periodic oscillations with constant amplitude over time for a given flight condition. It should be noted that limit cycle oscillations are undesirable since they can cause structural fatigue and pilot fatigue. Predicting the onset, amplitude and frequency of limit cycle oscillations (LCO) is an active research topic currently under development [55, 57]. An excellent review on nonlinear aeroelasticity can be found in [140]. Another related topic is the prediction of oscillations in nonlinear flight dynamics. When information on possibly unstable oscillations is provided to the pilot, certain controls may be activated in order to ensure a safe cruising performance. It is clear that the ability to accurately predict and identify the occurrence of nonlinear oscillations is of crucial importance in many scientific and engineering applications. Considerable efforts have been devoted to develop reliable methods for such predictions.

---

[1] Versions of some of the discussions in this chapter have been published in [238, 239, 240, 241] or submitted for publication in [241].

In the investigation of nonlinear aeroelastic behaviours, mathematical and computational techniques are frequently used. The describing function technique, the center manifold theory, and more recently the point transformation method have been successfully applied to predict limit cycle oscillations and other nonlinear responses of an aeroelastic system with structural nonlinearities. Numerical techniques based on the Houbolt finite-difference scheme and the Runge-Kutta time-integration procedure are also commonly used to study the nonlinear motions affected by structural nonlinearities [140, 151].

In these conventional approaches, a mathematical model is first constructed, and the behavior of the system under certain operating conditions is then predicted analytically or numerically as a solution in the form of a trajectory of a nonlinear dynamical system [140, 151]. In general, the mathematical model is represented as a set of partial differential and integral equations, and, in order for the model to be useful, the system parameters have to be given. If there are some uncertainties in developing the associated model or if some of the system parameters are not known, this approach will be impossible to implement. Many models are too complex to be solved analytically, therefore solving them numerically by computer implementation may be the only option. When modelling complex phenomena, computer simulations may take many days until a solution is determined, and the numerical algorithms used may not be robust due to errors in measurement or in estimation of the system parameters. In addition, numerical results can be incorrect if the algorithm fails to capture important features of the physical process modeled [151]. Moreover, in some applications, such as the ground vibration test or the actual flight test of an aircraft, only the dynamic response due to a given excitation is available. For such practical problems, the recorded nonlinear behaviors are noisy, non-stationary and have high dimensional dynamics. Hence, it is necessary to develop a prediction technique based only on the known response data rather than mathematically modelling the underlying physical phenomenon. Theoretical results suggest that, in principle, the positive limit set of a multidimensional trajectory can be reconstructed even by only using information about its projection on one of its dimensions [224].

In the present study, we propose to analyze the system dynamics based on the response data instead of using mathematical modelling and numerical simulations. The proposed data mining approach is based on a modern signal processing technique utilizing the power of artificial neural networks (ANNs). The objective of our work is to predict the long-term behaviour, in particular the asymptotic state, of a discrete-time nonlinear trajectory — which is usually one of the components of a trajectory of a multidimensional dynamical system — based on the information of a limited segment of its transient state. This amounts to predicting a set of future consecutive observations that is several times the size of the known data set. The ability to predict the asymptotic response based on the limited information of a given initial dynamics, such as data obtained from experiments, flight test or numerical simulation, could be important in practical applications.

The proposed approach begins with modelling the known data set in terms of a nonlinear mapping Φ that describes the dependence of each data point (observation) in the time series on a finite number of past consecutive observations. In other words, the time evolution up to the present moment of the quantity that is of interest to us is modeled in terms of an input-output relationship. An estimate of the first unknown data point can then be provided in terms of known past consecutive observations, using the mapping Φ, a process referred to as a *one-step prediction (OSP)*. If this estimate is further employed in Φ in order to generate an estimate for the next unknown data point, and so on, then we are dealing with a *multi-step prediction (MSP)* process. In our study, the predicted data set is considerably larger that the training set. Since predicted values — as opposed to actual observations — are applied repeatedly into the mapping Φ in order to generate the subsequent predictions, the main difficulty when performing a long-term MSP (LTMSP) is the step-by-step propagation of prediction errors [238, 239, 240, 259, 260]. In implementing this method, no knowledge is required about the nature or the intimate mechanisms of the phenomenon being investigated. It has been theoretically proven that the asymptotic dynamics of a trajectory of a finite-dimensional nonlinear dynamical system can always be reconstructed based on only one of its components, if the number of inputs of the mapping Φ is chosen to be sufficiently large [224].

In our study, Φ is chosen to be the mapping that expresses the output of an artificial neural network (ANN) in terms of its inputs. ANNs are information processing systems that originated from an alternative approach to problem solving, the so-called *connectionist approach*.

1.2 Connectionism Versus Symbolism

Human beings tend to capture the knowledge about the real world into a set of discrete semantic objects, or *symbols*, which they manipulate according to a set of formal rules. When confronted with a real-life problem, our usual approach is to translate the problem into a set of *concepts* connected in a certain manner, and then to formalize the path leading to the solution in terms of an *algorithm*, a well defined recipe made of chronologically ordered operations that have to be performed in order to obtain the desired result [85]. When solving certain problems that arise in science and engineering, it is common to translate this algorithm into a computer program. The exact series of steps to be performed has to be provided in advance to the computer, and the data has to be in a precise, non-noisy format. Moreover, there is a clear correspondence between the semantic objects and the machine hardware, in the sense that each object can be located in a memory cell. Therefore, if a few memory locations are destroyed, the algorithm will crash [85].

It has been noticed that certain tasks, such as recognizing handwritten characters or a person's face, cannot be formalized in the above *symbolic-algorithmic* manner, perhaps because intuitive knowledge cannot be captured in a set of discrete concepts which obey certain rules of grammar. Therefore, an alternative approach is necessary in order to solve these problems. The recently introduced *connectionist* approach is based on the idea that the functioning of the brain can be viewed as the result of parallel and distributed information processing performed by many interconnected simple subsystems — the neurons [85]. Hence the idea of creating information processing systems that mimic some of the architectural features of the brain in the attempt to achieve some of its processing capabilities [61, 85, 86, 163]. These are the so-called *artificial neural networks (ANNs)*.

An ANN consists of interconnected simple processing elements, called *artificial neurons*, each of which receives several real numbers as inputs and computes a single output. The output is determined by a nonlinear function (*transfer function*) of a weighted sum of the inputs. The strength of an ANN is provided by the arrangement of neurons and the manner in which they are interconnected (the *network architecture*). The neurons are usually arranged in successive layers (*multi-layer feed-forward (MLFF)* architecture), such that the outputs of all neurons in one layer are provided as inputs to each neuron of the next layer. The inputs to all neurons in the first layer form the network input, while the outputs of all neurons in the last layer form the network output. The last layer of neurons is called the output layer, while all the other layer are generally referred to as hidden layers. The ANN output depends on the network inputs as well as the inter-neuron connection strengths, or *weights*. They can be determined by a process of learning (or *training*) from a set of examples of correct network outputs to given inputs (the *training set*). The most common type of ANN training involves minimizing the mean-square ANN output error over the training set using a nonlinear optimization procedure [86]. Once trained, the ANNs are capable of generalizing, that is, they can provide correct network outputs for network inputs never experimented before. Neural networks are robust in the presence of noise and hardware degradation [85]. Small changes in the ANN input or in a weight will not dramatically affect a neuron's output. Moreover, in an ANN there is no simple correspondence between neurons and semantic objects. Rather, the information corresponding to a semantic object is distributed throughout the network. It has been said, therefore, that ANNs "operate at a *sub-symbolic* level" [85].

Our choice of the mapping Φ is motivated by the proven fact that, in principle, any nonlinear function can be approximated with any desired accuracy by the output of a two-layer feed-forward ANN with a sigmoidal transfer function in the first layer (a 2LFF1SNN) and a linear transfer function in the second layer (a 2LFF1S2LNN), provided that sufficiently many neurons are available in the network's hidden layer [31, 32, 33, 34, 53, 75, 77, 76, 96, 97, 105]. A *sigmoidal function* is generally defined as a bounded function that has horizontal asymptotes at both $-\infty$ and $+\infty$, with the left-hand asymptote being lower than the right-hand asymptote. The universal approximation property still holds for 2LFF1SNNs with continuous and monotone sigmoidal transfer function in the output layer (2LFF1S2SNNs) [27].

1.3 Applications of Neural Networks

The universal approximation property of ANNs basically states that a neural network can accurately interpolate any nonlinear function based on a set of known values of the function. Therefore the most common applications of ANNs comprise nonlinear function approximation based on samples [29, 35, 40, 42, 64, 70, 82, 87, 99, 103, 117, 152, 162, 187, 223, 236, 261]. Moreover, ANNs are capable of approximating nonlinear mappings of complicated form arising in real-life situations and which cannot be determined analytically, such as the variation of the blood pressure of a patient as a function of his heart rate and his corporal acceleration [98], the chemical oxygen demand in a certain area (used as an index of water pollution) as a function of water temperature, transparency, and dissolved oxygen concentration [115], etc. ANNs have been used for identification of parameters in aerodynamic models [149], as well as in a radar point source location problem to determine the position of a source in the scene given the outputs, possibly corrupted by noise, of the array of receivers [253].

A particular type of function interpolation is the so-called *pattern classification* problem [4, 6, 13, 64, 114, 191, 254]. In that case, the network input is an encoding of the object to be classified, and the single ANN output represents the class to which the input pattern belongs. ANNs have been used for classification of nonlinear trajectories [102], mushrooms classification into edible and poisonous based on 22 features [127], classification of Iris flowers into three categories (subspecies) based on four or seven features [72, 217], diagnosis of heart diseases based on analysis of the electrocardiographic signal [54, 62], diabetes diagnosis (positive/negative) [59, 104, 227], breast cancer diagnosis (benign/malignant lumps) based on nine cytological features of the breast lump [54, 104, 136, 214], spike detection in epileptic electroencephalographic signals [59], discrimination between mines and rocks in sonar images [237], etc.

A benchmark pattern classification for ANNs is the well-known *XOR problem*, whose difficulty lies in the fact that there is no contiguity relationship between samples in the same class [5, 11, 12, 18, 54, 82, 120, 141, 184, 249]. The *parity problem* is an extension of the XOR problem, in which the ANN input is a string of 0's and 1's and the output is 1 if the input contains an odd number of 1's and 0 otherwise [26, 54, 58, 83, 108, 157, 205, 214, 247, 252, 262, 265]. In the *two spirals problem*, the task is to correctly classify two sets of training points that lie on two distinct spirals which twist three times around the origin and around each other in the plane [45, 134, 227, 256, 265].

An application of ANNs in experimental high-energy physics consists in discriminating between patterns of collision in the large electron-positron collider into two classes: 'background noise' or 'potentially relevant event' [12]. ANNs have also been used for prediction of developing a postoperative spinal deformity called kyphosis, based on the age of the child, the number of vertebrae involved in the spinal operation, and the beginning of the range of the vertebrae involved in the operation [233]. In economics applications, ANNs were proposed to detect target companies of the Securities and Exchange Committee's investigation of fraudulent financial reporting [135] or to predict the failure or survival of a set of credit unions based on five financial variables [134].

Many pattern classification applications can be formulated as *pattern recognition* problems. ANNs have successfully recognized typed numerical digits [120, 145, 262] and letters [58, 104, 106, 116, 171, 184, 205, 237], as well as handwritten numerical digits [60, 106, 180, 189] and characters [145, 236]. This type of application is also referred to as character *encoding*. Neural networks have also been used in different speech recognition applications [137], such as recognition of a spoken vowel [237], word [127], or speaker identification [247]. Other applications include recognition of faulty LED-display digits [127] or of promoters in DNA nucleotide strings [72], encoders, adders, multiplexers, demultiplexers [54, 184, 205], etc.

ANNs have recently been used in aerospace industry applications, such as damage detection [263], aerodynamic design [197], estimation of air-data parameters [201], detection of airframe ice based on the dynamic response of the aircraft to known elevator inputs [111], estimation of the strain on the vertical tail structure as a function of the lateral and normal acceleration measured on several points on the empennage during various flight maneuvers [121], etc.

Automatic control devices are widely used in various machines, ranging from household appliances to space shuttles. The goal is to manipulate a system so that it behaves in a desired fashion. When designing a controller, an important preliminary step is to understand how the system will respond to various stimuli. A mathematical model of the system has to be constructed. This can be achieved either in a deductive manner, by studying in detail the physical process involved, or can be inferred from a set of input-output data collected during a practical experiment with the given system. The second approach is referred to as *system identification* [179].

Neural networks have been widely used in various nonlinear system identification applications [3, 7, 28, 40, 175, 181, 251]. The typical setting is that of a discrete-time system [46, 155, 186, 196, 220, 249], given the fact that measurement of the inputs and outputs can only be taken at discrete time moments. ANNs may succeed where mainstream methods fail, especially in the case of the so-called *arbitrarily nonlinear systems* [210], which do not have a finite-order polynomial expansion. Therefore it is very difficult, if not impossible, to identify them using nonlinear models based on finite-order polynomial expansions, such as the truncated Volterra series [52, 67, 132, 169, 267]. Neural nets have been used for approximation of stochastic processes [234], for modelling the dynamics of a hydraulically controlled robot arm [93] or of processes in a chemical reactor [266], such as distillation [272], etc.

Sometimes system identification is implemented as a component of the controller. This type of applications is known as *adaptive control* [143, 144, 172, 173, 174], and is typically designed for systems whose dynamics varies with time. By using the universal approximation capability, the adaptive controller based on ANNs can be designed without explicit knowledge of the system dynamics [1, 70, 129, 195, 208, 219, 226]. Applications of ANNs include nonlinear system identification and adaptive control of mathematically simulated plants [150, 211], of anesthesia and muscle relaxation [15], of processes inside a bioreactor [195], of a boat [192], and of a truck-and-trailer backup system [12].

ANNs have recently been applied in flight control problems [69, 139, 146]. Applications in helicopter stabilization [65], adaptive control of antiair missiles [165], and active control of aeroelastic response [16, 213] have been reported. The latter has the purpose of designing flutter suppression control laws. A variation of adaptive control is *adaptive filtering*, which is concerned with reducing the additive noise from a corrupted signal [153, 192]. Simulation results show that the neural filters with only a few hidden neurons consistently outperform the extended Kalman filter for the simple nonlinear signals being investigated in [153].

The time series prediction capabilities of ANNs have been investigated by many researchers. In most applications, the dependence of each data point x_{t+1} of a nonlinear time series on a finite number of past consecutive observations x_t, x_{t-1}, …, x_{t-n_0+1} is modeled using a mapping $\Phi_{\mathbf{w}}$ given by the output of a neural network. This mapping is then used to estimate unknown observations, taking only known data points as inputs. This process is known as *one-step prediction (OSP)*.

Various benchmark nonlinear time series have been used to investigate the OSP capabilities of neural nets. The *sunspots data* set represents the time series of the average sunspot numbers recorded every year since 1770 [9, 25, 51, 141, 177, 218, 255, 261, 269]. The *Canadian Lynx time series* is a record of the annual fur returns at auction in London by the Hudson Bay Company between 1821 and 1934 [269]. OSP of Brownian motion [107] and other mathematically generated nonlinear time series [7, 14, 47, 167], especially chaotic time series [44, 45, 177], has also been performed using ANNs. Of particular interest are OSP applications for the Mackey-Glass time series [9, 38, 41, 62, 66, 94, 162, 166, 198, 209, 218, 261], for the Hénon map [9, 41, 131], and for the Lorenz-type time series [9, 66, 166, 202].

Economics applications include OSP of different financial time series [270], such as the raw trading volume on the New York Stock Exchange [138], the Dow Jones Industrial Average [138, 177], the Korean stock market index [122], the stock trend for major companies [117, 207], the foreign currency exchange rate [25, 36, 264, 269], and the unemployment rate [90].

In meteorology, ANNs have been used for OSP of the temperature distributions in various locations around the world [74], the volume of rainfall in a certain region [115], different features of solar activity [48], and the El Niño southern oscillation phenomenon, i.e., the sea surface temperature anomaly, the zonal pseudo wind stress anomaly, and the meridional pseudo wind stress anomaly [156].

An important application of ANNs is in *short-term load forecasting (STLF)* [41, 47, 51, 119, 221], which deals with predicting electric loads (or, in other words, electric power demands) for a period of hours, days, or weeks, in order to adapt energy generation to energy demand. STLF plays a significant role in the real-time control and the security functions of an energy management system. Forecast error in load predictions results in increased operation costs.

Other OSP applications involve speech signals [10, 91, 159, 160, 158, 232], building electricity and water usage in an institutional building [25], measurements from a Laser system [25, 177], the external sound level of an automatic transmission system with application in fault detection [2], etc.

When forecasting observations further away in the future is necesary, a *multi-step-ahead prediction (MSAP)* process is used in most applications. In a MSAP process, the data point x_{t+d}, situated d steps ahead in the future, is predicted directly based on a finite number of present and past consecutive observations x_t, x_{t-1}, …, x_{t-n_0+1}, using a mapping $\Phi_{\mathbf{w}}$ given by the output of a neural network. An MSAP process can only be used for short-term time series prediction. As the lag d increases, the dependence of x_{t+d} on the present and past consecutive observations becomes more complex, and thus more difficult to model based on the limited available data set.

ANNs have been used to perform MSAP for various mathematically generated nonlinear time series [188, 272], such as the Mackey-Glass time series [19, 38, 115, 122, 166, 202], the Lorentz time series [166], etc. MSAP of a nonlinear time series that models a steam generator (widely encountered in nuclear power plants) [188] has also been reported. Short-term MSAP of the traffic volume has been performed, with applications in traffic flow control [123].

ANNs have been applied for MSAP of the sunspot time series [78, 203, 244] and the Nile flow level (cubic metres/day) [8]. The latter is one of the longest recorded time series of a natural phenomenon. It has been noted that preprocessing such as differencing the output (which accentuates the noise), subtracting the seasonal average, or taking the discrete Fourier series, could lead to worse results than the basic method with no preprocessing [8].

In a so-called *multi-step prediction (MSP)* process, the data point x_{t+d} is predicted based on a finite number of present and past consecutive observations, in a sequence of d steps. A mapping $\Phi_{\mathbf{w}}$, given by the output of an ANN, first provides an estimate (prediction) $\hat{x}_{t+1}$ of the unknown data point x_{t+1}, taking as inputs the known values x_t, x_{t-1}, …, x_{t-n_0+1}. The values $\hat{x}_{t+1}$, x_t, x_{t-1}, …, x_{t-n_0+2} are then used as inputs in $\Phi_{\mathbf{w}}$ in order to generate a prediction $\hat{x}_{t+2}$ for x_{t+2}. By repeating the process d times, an estimate $\hat{x}_{t+d}$ of x_{t+d} is eventually computed.

The reported MSP applications of ANNs are very few, and are only concerned with short-term prediction of time series. Short-term MSP of the Nile flow level [8], the sunspots time series [255], and of chaotic time series [44], such as the Mackey-Glass [62, 166] or Lorentz time series [166], has been reported. However, we have no knowledge of any long-term MSP application of ANNs so far.

Nonlinear dynamical systems applications of ANNs for small-scale problems have also been reported. ANNs have been used for learning continuous-time oscillatory trajectories [178, 206], or in the trajectory generation problem (designing a dynamical system whose terminal behavior emulates a prespecified spatio-temporal pattern independently of its initial conditions) [268] or trajectory modulation problem (designing systems that control the trajectory generation process by means of external inputs) [268]. Various other uses of ANNs in solving practical problems include image processing applications [118, 216], such as image filtering and segmentation [162], binary image storage [109], grammatical inference [63, 147, 148, 248], approximately solving linear systems of algebraic equations in real time [43], etc.

1.4 Original Contributions

The research presented in this study is an interdisciplinary one, and it covers several major fields such as mathematics, statistics, computing science and engineering. Our problem focuses on the LTMSP of nonlinear time series. The field of nonlinear time series analysis is still a poorly developed research field, most of the work having been done since the 1980's [68]. The general field of time series analysis has been dealing mainly with linear models, which are now well understood [215]. For nonlinear models, however, theoretical analysis is difficult, hence their understanding is in general incomplete. In addition, time series forecasting mainly deals with OSP, due to the theoretical tractability of this process. To predict data points further in the future, MSAP is usually the method of choice. Though still theoretically tractable, this method requires a much more complicated analysis [30, 84, 95, 113, 176, 228], and the results obtained are not suitable for practical use if longer-term prediction is required. The so-called *one step plug-in method*, or MSP (in our nomenclature), is considered to be undesirable, due to the step-by-step error propagation and to the difficulty of performing any theoretical analysis. Therefore long-term prediction of nonlinear time series has not been well studied.

The field of ANNs is very new, most of the work having been done since 1990. The overwhelming majority of ANN applications are dealing with nonlinear function approximation, pattern classification, and OSP of time series. The common feature of these applications is that they can be formulated as interpolation problems. Namely, an ANN is used to approximate a nonlinear function on a certain domain by generalizing from a set of function values. Most of the theoretical analysis of ANN behavior and most of the tools in the ANN field have been designed for interpolative applications. Very few authors have attempted to perform MSP using ANNs, and most published results are limited to short-term prediction. Long-term MSP cannot be formulated as an interpolation problem, and new tools are needed in order to approach this problem. Redundancy in the ANN parameters is much more important in this new context, and the additional problems of insufficient training data and recursive error propagation make LTMSP a very difficult task. To the best of our knowledge, our work is one of the first extensive investigations in this new research direction.

The idea of using ANNs to forecast the asymptotic behavior of nonlinear dynamics is quite new. To predict the frequency and amplitude of the LCO of an aircraft wing, Denegri and

Johnson (2001) [57] trained a three-layer ANN to output these two values when given the Mach number of the desired flight condition, and the values of some parameters resulted from the linear flutter analysis [55, 56]. Our method (whose development started in 1999) uses a different approach, namely training an ANN based on the known transient data set and subsequently using the neural network to reconstruct the asymptotic state by a MSP process. Our approach to LTMSP using ANNs has been successful for a certain class of problems. We have designed several novel ANN architectures, with features that control the error propagation in the MSP process, which have proven to be capable of providing accurate predictions of nonlinear oscillatory motions arising in aeroelasticity [238, 239, 240, 259, 260].

The practical implementation of ANNs for a given problem is not as straightforward as the elegant theoretical results [31, 32, 33, 34, 53, 75, 76, 77, 96, 97, 105] might suggest. Cybenko [53] notes that the number of neurons necessary in most practical problems is most likely astronomically high, due to the so-called curse of dimensionality. Using many neurons is computationally expensive to the point that solving the problem may become unfeasible. Hornik [96] observes that not all transfer functions that satisfy the theoretical requirements for the universal approximation will perform equally well in practical problems. Due to the complicated shape of the error surfaces, the ANN training often converges to a local minimum, which may not be a solution of the given problem. Since in MSP applications the use of global optimization algorithms (genetic, annealing) is prohibitive because of the large number of parameters (ANN weights) to be estimated, one is forced to employ point-by-point nonlinear optimization algorithms, which depend heavily on the initial guess on the weight values. How to determine good initial values of the network weights for ANN training is still an unsolved problem.

Given these practical difficulties, we are interested in developing a technique for extracting the maximum information from the training set using a minimum number of hidden neurons. This would reduce the training time and provide better generalization capabilities and robustness in the presence of noise, as a consequence of having fewer degrees-of-freedom in the system. The successful ANN architectures previously proposed [238, 239, 240, 259, 260] involved either some kind of weight scaling or using a scaled sigmoidal transfer function in the output layer. In the present study we investigate the effect of these two architectural features on the ANN MSP performance under neuron scarcity conditions. Various ANN architectural features, weight initialization procedures, and different choices of learning rate are compared. Our investigation presented here has clearly demonstrated that the ANN training using an adaptive learning rate often does not converge when the training set is noisy, and therefore a constant learning rate should be used. Initializing the first-layer weights with normalized segments of the training set has proved to lead to a much better prediction accuracy than a random weight initialization, when few hidden neurons are used. Moreover, normalizing the second-layer weights provides great robustness in the presence of noise. This constitutes a significant architectural and training alteration of the classical 2LFF1S2LNN typically used in interpolative-type applications (including OSP and MSAP). Indiscriminately applying the standard ANN methodology to LTMSP applications leads to a poor prediction performance.

How to choose an appropriate number of ANN inputs and neurons for a given application is a crucial issue in the field of ANNs. Choosing the number of inputs is quite straightforward in the context of pattern recognition and function interpolation in general, where the input and output spaces have a clear physical meaning. In OSP or MSAP, typically a few inputs are sufficient in order to provide an accurate prediction, since no error propagation is involved. In the case of MSP, we expect that more redundancy is needed in order to compensate for the recursive error propagation. In order to estimate the optimum number of neurons in an ANN, various empirical formulas [61] and criteria based on information theory [61, 71, 170] have been proposed. However, all these estimates have been developed in the context of interpolative-type applications, and therefore are not relevant to our MSP problem. We have proposed methods for consistently choosing appropriate values for the ANN inputs and hidden neurons in a given application of the type considered in tis study [240]. A stopping criterion for the ANN training, specific to this type of applications, has also been designed [238, 239].

The developed neural networks have been tested on both real-life experimental data and numerically simulated data sets describing the oscillatory motions of a two-degree-of-freedom nonlinear aeroelastic system. Once a long-term prediction of a trajectory is achieved, different features, such as damping and frequency components, may be extracted from the predicted signal, using an ANN in conjunction with a wavelet decomposition module. We developed a feature extraction method that greatly improves the computational efficiency of the ANN module in such applications [238, 239]. We were also able to correctly predict the dynamic interaction in a complex cavity flow problem, but further research is needed to improve the accuracy of these predictions.

This thesis is organized as follows. An overview of ANN architectures, training algorithms, and the universal approximation results, is presented in Chapter 2. In Chapter 3, the main research contribution of this study is presented. A review of time series analysis methods is provided in Section 3.1, followed by a detailed presentation of the proposed approach to long-term prediction of nonlinear oscilaltions using ANNs (in Section 3.2). The implementation of the proposed method is presented in Section 3.3, while in Section 3.4 a rudimentary theoretical justification of the proposed approach is discussed. More specifically, it is proven that, under certain conditions, the asymptotic state of the ANN-generated long-term prediction is close to the asymptotic state of the trajectory to be predicted. In Chapter 4, 12 combinations of architectural and training algorithm features are compared based on 8 test cases, in order to assess which combination robustly extracts the most information from the known data set using the minimum number of neurons. The stability of the best two methods with respect to variations in the number of inputs, hiddden layer neurons, and training data points is investigated. Further applications of the proposed approach are reported in Chapter 5. Conclusions and future research directions are provided in Chapter 6.

Chapter 2

Artificial Neural Networks (ANNs)

2.1 Mathematical Representation of ANNs

In a biological neuron (see Fig.2.1), the electrical signals are carried through the *dendrites* into the *cell body*, where they are summed and thresholded, and the resulting signal is sent through the *axon* out to a dendrite of another cell [86]. Similarly, an artificial neuron (see Fig.2.2) receives the inputs $x_1, x_2, \ldots, x_{n_0}$ (real numbers), which are weighted by $w_1, w_2, \ldots, w_{n_0}$ (the neuron's *weights*), then a *bias* w_0 is added, resulting in the *net input* $v = w_0 + w_1x_1 + \ldots w_{n_0}x_{n_0}$. Note that the bias can be viewed as a particular weight corresponding to a constant input equal to 1. For simplicity, the term "weights" will be used for w_0 as well as for $w_1, w_2, \ldots, w_{n_0}$.

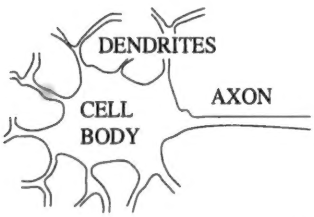

Figure 2.1: Biological Neuron.

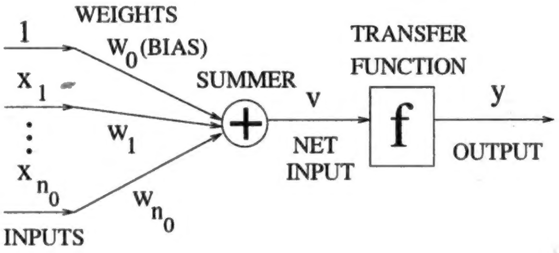

Figure 2.2: Artificial Neuron.

A *transfer function (TF)* $f(\cdot)$ performs the thresholding and yields the *output* $y = f(v)$ of the artificial neuron. The transfer function is chosen according to the requirements of the problem that has to be solved. Common transfer functions used in applications are $f(v) =$

v, $f(v) = \tanh(v)$, $f(v) = [1 + \exp(-v)]^{-1}$, $f(v) = H(v)$ (Heaviside's unit step function), $f(v) = \ln(1 + |v|)\operatorname{sgn}(v)$ [61, 86], staircase functions [216], cubic spline functions [236], hysteresis-type functions [250], etc. Parameter-dependent transfer functions can also be used, for instance $f_{\alpha,\beta}(v) = \alpha\tanh(\beta v)$ (where α, β represent the amplitude and slope of the sigmoid, respectively) [49, 230].

The artificial neuron described above is the classical type of neuron proposed by McCulloch and Pitts [164] in their 1943 paper that laid the foundation for the field of artificial neural networks. Since then, various types of artificial neurons have been introduced, many of them representing significant alterations of the classical architecture. For instance, Cotter [50] proposed an artificial neuron of the type:

$$y = \left\{ 1 + \sum_{l_0=1}^{m_0} \exp\left(- \sum_{k_0=1}^{n_0} w_{l_0,k_0} x_{k_0} \right) \right\}^{-1}, \tag{2.1}$$

due to the universal approximation property of linear combinations of such mappings, according to the Stone-Weierstrass theorem. Chiang and Fu [37] proposed a so-called *multi-threshold quadratic sigmoidal neuron*, for which the output

$$y = \left\{ 1 + \exp\left[\left(w_0 + \sum_{k_0=1}^{n_0} w_{k_0} x_{k_0} \right)^2 - \left(\theta_0 + \sum_{k_0=1}^{n_0} \theta_{k_0} x_{k_0} \right) \right] \right\}^{-1} \tag{2.2}$$

depends on the weights $w_0, \ldots, w_{n_0}$ as well as on the *thresholds* $\theta_0, \ldots, \theta_{n_0}$. There has also been considerable interest in experimenting with the so-called *high-order* neurons, in which the net input v is a polynomial of degree larger that one in the inputs $x_1, \ldots, x_{n_0}$ [24, 83, 126, 127, 177, 189, 200, 254, 270, 271]. Most of these neuron architectures cannot be justified biologically anymore, but they are rather serving some specific purposes. We have studied various types of neurons, but have not yet observed any significant performance improvement compared to the classical neuron architecture. Therefore, in accord with the parsimony principle (everything else being equal, a simple model is usually preferable [68]), we have mainly used the classical neuron structure in our study.

The neurons are connected to each other in various ways. Each neuron can receive inputs from outside the network, from other neurons, or from itself. The output of a neuron can be fed as input to other neurons, or to itself, or to the outside. Each connection has its corresponding strength, or *weight* . A widely used ANN architecture is the *multi-layer feedforward (MLFF)* architecture, in which the neurons are arranged in successive layers. The outputs $y_h^{(l)}$, $1 \le h \le n_l$ of all neurons in each layer l are fed as inputs to each neuron in the next layer $l+1$. Thus, the output of the k-th neuron in layer $l+1$ $(1 \le k \le n_{l+1})$ has the expression

$$y_k^{(l+1)} = f^{(l+1)}\left\{ v_k^{(l+1)} \right\} = f^{(l+1)}\left\{ w_{k,0}^{(l+1)} + w_{k,1}^{(l+1)} y_1^{(l)} + \ldots w_{k,n_l}^{(l+1)} y_{n_l}^{(l)} \right\} \tag{2.3}$$

where $w_{k,0}^{(l+1)}, \ldots, w_{k,n_l}^{(l+1)}$ are the weights of that neuron. Traditionally, for each layer l, the same transfer function $f^{(l)}(\cdot)$ is applied to all neurons in that layer. The inputs $x_h = y_h^{(0)}$, $1 \le h \le n_0$

C

to all neurons in the first layer (forming the *network input* $\mathbf{x} = \mathbf{y}^{(0)}$) are propagated forward through the successive layers of neurons. The outputs $y_h = y_h^{(L)}$, $1 \le h \le n_L$ of all neurons in the last layer L (the *output layer*) are considered to form the *network output* $\mathbf{y} = \mathbf{y}^{(L)}$. Clearly, the network output depends on the ANN input as well as on the values of all network weights (stored into the vector $\mathbf{w}$): $\mathbf{y} = \mathbf{y}(\mathbf{x}, \mathbf{w})$. A two-layer ($L = 2$) feedforward ANN (a 2LFFNN) is represented in Fig.2.3.

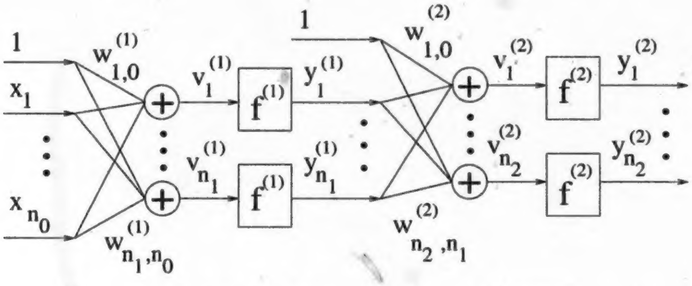

Figure 2.3: Two Layer Feedforward ANN.

There is a wide variety of ANN architectures [61, 86]. Here we discuss several important examples. For instance, some authors have proposed ANNs with architectures inspired from the classical mathematical and computational methods. Cotter [50] proposed architectures that implement classes of functions for which the Stone-Weierstrass density theorem is applicable, while Wang and Lin [251] and Tsitouras [231] proposed MLFF ANNs performing the Runge-Kutta time-marching procedure.

Recently there has been considerable interest in the field of *radial basis function (RBF) networks* [2, 19, 38, 39, 45, 81, 146, 210, 212, 217, 225]. The output of a RBF neuron that receives the input $\mathbf{x} = [x_1, \ldots, x_{n_0}] \in \mathbf{R}^{n_0}$ is defined by $y = \psi(||\mathbf{x} - \mathbf{c}||_2)$ ($|| \cdot ||_2$ is the Euclidean norm), where $\mathbf{c} \in \mathbf{R}^{n_0}$ is a so-called *center*, and $\psi : [0, \infty) \to [0, \infty)$ is usually a continuously differentiable decreasing function. The most commonly used functions are $\psi_\alpha(x) = \exp(-x^2/\alpha^2)$ (Gaussian function), $\psi_\alpha(x) = (\alpha^2 + x^2)^{-1/2}$ (inverse multiquadratic function), B-spline functions [210], $\psi(||\mathbf{x}||_2) = (n_0 - ||\mathbf{x}||_2^2)\exp(-||\mathbf{x}||_2^2/2)$, for $\mathbf{x} \in \mathbf{R}^{n_0}$ (the Mexican Hat function, very popular in wavelet applications) [94], etc. It has been proved that the output of a two-layer RBF network with linear transfer function in the output layer (and a single output neuron)

$$y = \sum_{k_1=1}^{n_1} w_{k_1} \psi(||\mathbf{x} - \mathbf{c}_{k_1}||_2) \tag{2.4}$$

can approximate any nonlinear function of n_0 variables with any accuracy, provided that sufficiently many RBF neurons (that is, sufficiently many centres) are available.

In certain types of applications, such as system identification, adaptive control, time series prediction, etc, the ANN receives as input a vector $\mathbf{x}(t) \in \mathbf{R}^{n_0}$ and generates the output $\mathbf{y}(t) \in \mathbf{R}^{n_L}$, at each discrete time step $t \in \mathbf{Z}_+$. If the output of at least one neuron at time t is fed as one of the inputs of some neuron in a previous (or in the same) layer, at a later time $t+d$, then the ANN belongs to the class of *recurrent networks*. An important feature of the network is the existence of a *feedback connection*.

A classical example of a recurrent ANN is the Elman network [63, 128], which is a 2LFFNN to which feedback connections from the hidden layer to the input have been added, in the sense that, at each time step t, every neuron in the hidden layer receives as inputs both the external input $\mathbf{x}(t)$ and the outputs of all hidden neurons at step $t-1$, i.e., the vector $\mathbf{y}^{(1)}(t-1)$. Thus, the output of each hidden neuron at step t has the expression:

$$y_{k_1}^{(1)}(t) = f^{(1)}\left\{w_{k_1,0}^{(1)} + \sum_{k_0=1}^{n_0} w_{k_1,k_0}^{(1)} x_{k_0}(t) + \sum_{l_1=1}^{n_1} w_{k_1,n_0+l_1}^{(1)} y_{l_1}^{(1)}(t-1)\right\} \tag{2.5}$$

for $1 \leq k_1 \leq n_1$. A particular Elman-type ANN for which $w_{k_1,n_0+l_1}^{(1)} = 0$ except when $l_1 = k_1$ (i.e., each hidden neuron only has a feedback connection with itself) is the *diagonal recurrent neural network (DRNN)* [129]. This network belongs to a class of reccurrent ANNs called the *locally recurrent globally feedforward (LRGF)* networks, which are MLFF networks for which feedback connections from each neuron to itself are added. For instance, the neuron proposed by Frasconi, Gori, and Soda [232] generates its output $y(t)$ by receiving the external vectors $\mathbf{x}(t)$, $\mathbf{x}(t-1)$, …, $\mathbf{x}(t-d)$ and its own past outputs $y(t-1)$, …, $y(t-d)$, as inputs:

$$y(t) = f\left\{\sum_{k=1}^{n_0}\sum_{h=0}^{d} w_{k,h} x_k(t-h) + \sum_{h=1}^{d} w_{n_0+1,h}\, y(t-h)\right\}. \tag{2.6}$$

If $w_{n_0+1,h} = 0$, $\forall h$, we obtain the output of a *time-delay neuron*. Time-delay neural networks (TDNN) [46, 122, 137, 243, 245, 246] are not LRGFNNs. The neurons do not have recurrent connections, but only a memory of past inputs they received.

Poddar and Unnikrishnan [211, 232] proposed a LRGFNN where to each classical-style neuron is attached a *memory neuron*. At each moment t, the output $y_k^{(l)}(t)$ of the k-th neuron in the l-th layer ($1 \leq k \leq n_l$) and the output $z_k^{(l)}(t)$ of its attached memory neuron have the expressions:

$$\begin{aligned} y_k^{(l)}(t) &= f^{(l)}\left\{\sum_{j=1}^{n_{l-1}} w_{k,j}^{(l)} y_j^{(l-1)}(t) + \sum_{j=1}^{n_{l-1}} w_{k,n_{l-1}+j}^{(l)} z_j^{(l-1)}(t)\right\} \\ z_k^{(l)}(t) &= \alpha_k^{(l)} y_k^{(l)}(t-1) + (1-\alpha_k^{(l)}) z_k^{(l)}(t-1) \end{aligned} \tag{2.7}$$

where $\alpha_k^{(l)} \in [0,1]$, $\forall l, k$. There are many other types of ANN architectures (self-organizing maps [86, 111], Hopfield ANNs [17, 86, 112], Grossberg ANNs [86]), but we are not concerned with them in the present study.

2.2 ANN Training

One of the main issues when working with neural networks is to determine acceptable values for the weight parameters such that the network can successfully perform a specific task. A neural network "learns" to perform a certain task during the process called *network training.* In general, an initial guess of the weight values is first set, then the weights are iteratively updated according to a certain formula called the *learning rule* [86], until a given *stopping criterion* is met. There are several types of neural network learning. In a *supervised learning*, the network is provided with a set of examples of network inputs and the corresponding correct outputs (*target outputs*), called *training set*: $\{\tilde{\mathbf{x}}(1), \tilde{\mathbf{y}}(1)\}$, $\{\tilde{\mathbf{x}}(2), \tilde{\mathbf{y}}(2)\}$, ..., $\{\tilde{\mathbf{x}}(M), \tilde{\mathbf{y}}(M)\}$. In a *reinforcement learning* process, for each of the sample inputs $\tilde{\mathbf{x}}(1)$, $\tilde{\mathbf{x}}(2)$, ..., $\tilde{\mathbf{x}}(M)$, no target output is given, but the network receives a grade (a measure of its performance) for its corresponding output. In an *unsupervised learning*, the weights are modified according to the network inputs only (no target outputs are given), by performing a certain *clustering* operation on the set of input patterns [86]. Since the target outputs to sample inputs are available in our study, the ANN learning that is relevant to our study is the supervised learning.

Among the supervised learning rules, the most important one is the *performance learning* method, in which the network weights are iteratively adjusted in order to optimize the network performance. Let the mean squared output error over the training set:

$$E(\mathbf{w}) \overset{\text{def}}{=} \frac{1}{M}\sum_{m=1}^{M} ||\mathbf{e}(m,\mathbf{w})||_2^2 = \frac{1}{M}\sum_{m=1}^{M} ||\tilde{\mathbf{y}}(m) - \mathbf{y}(\tilde{\mathbf{x}}(m),\mathbf{w})||_2^2 \tag{2.8}$$

define the *performance index*, which is small when the network performs well and large when the network performs poorly. In the above equation, $\mathbf{y}(\tilde{\mathbf{x}}(m),\mathbf{w})$ is the network output corresponding to the input $\tilde{\mathbf{x}}(m)$ and weights $\mathbf{w}$, and $\mathbf{e}(m,\mathbf{w}) = \tilde{\mathbf{y}}(m) - \mathbf{y}(\tilde{\mathbf{x}}(m),\mathbf{w})$ is the output error corresponding to the m-th training pattern. $E(\cdot)$ is a nonnegative functional on $\mathbf{R}^{n_{\mathrm{w}}}$, where n_{w} is the number of ANN weights.

Alternative error measures have been used by some authors, for instance

$$E(\mathbf{w}) \overset{\text{def}}{=} \frac{1}{M}\sum_{m=1}^{M} L(\tilde{y}(m), y(\tilde{\mathbf{x}}(m),\mathbf{w})), \tag{2.9}$$

for an ANN with a single output, contained in $[0,1]$, where $L : [0,1] \times [0,1] \to \mathbf{R}$

$$L(x,y) \overset{\text{def}}{=} x\ln\left(\frac{x}{y}\right) + (1-x)\ln\left(\frac{1-x}{1-y}\right) \tag{2.10}$$

is known as the *entropic loss* (where, by definition, $0\ln(0) = 0\ln(0/0) = 0$) [92, 180].

The minimization of the performance index can be performed by using any nonlinear optimization algorithm. The most widely used optimization methods are the gradient-based, point-by-point algorithms. In a MLFFNN, the gradient $\nabla E(\mathbf{w})$ of the mean-squared error (2.8) is usually computed by the so-called *backpropagation algorithm.* If $\{\tilde{\mathbf{x}}(1), \tilde{\mathbf{y}}(1)\}$, $\{\tilde{\mathbf{x}}(2), \tilde{\mathbf{y}}(2)\}$, ..., $\{\tilde{\mathbf{x}}(M), \tilde{\mathbf{y}}(M)\}$ is the training set, we define:

$$\mathbf{y}^{(0)}(m) \overset{\text{def}}{=} \tilde{\mathbf{x}}(m), \quad y_0^{(l)}(m) \overset{\text{def}}{=} 1, \quad 1 \le m \le M,\ 1 \le l \le L. \tag{2.11}$$

First, the ANN inputs are 'propagated forwards', namely for $1 \le k \le n_L$, $1 \le l \le L$, we have:

$$v_k^{(l)}(m) \stackrel{\text{def}}{=} \sum_{j=0}^{n_{l-1}} w_{k,j}^{(l)} y_j^{(l-1)}(m), \quad y_k^{(l)}(m) \stackrel{\text{def}}{=} f^{(l)}(v_k^{(l)}(m)), \quad z_k^{(l)}(m) \stackrel{\text{def}}{=} \dot{f}^{(l)}(v_k^{(l)}(m)). \tag{2.12}$$

For simplicity, we write $\mathbf{e}(m)$ instead of $\mathbf{e}(m, \mathbf{w})$ in (2.8) and we define

$$E(m) \stackrel{\text{def}}{=} ||\mathbf{e}(m)||_2^2 = \sum_{h=1}^{n_L} |e_h(m)|^2 = \sum_{h=1}^{n_L} |\tilde{y}_h(m) - y_h^{(l)}(m)|^2. \tag{2.13}$$

Then the output errors are 'propagated backwards':

$$\delta_h^{(L)}(m) \stackrel{\text{def}}{=} \frac{\partial E(m)}{\partial v_h^{(L)}(m)} = -2 z_h^{(L)}(m) e_h(m), \quad 1 \le h \le n_L, \tag{2.14}$$

$$\delta_k^{(l)}(m) \stackrel{\text{def}}{=} \frac{\partial E(m)}{\partial v_k^{(l)}(m)} = z_k^{(l)}(m) \sum_{h=1}^{n_{l+1}} w_{h,k}^{(l+1)} \delta_h^{(l+1)}(m), \quad 1 \le k \le n_l, \ 1 \le l \le L-1. \tag{2.15}$$

The gradient components are then computed by the formul[illegible]

$$\frac{\partial E}{\partial w_{k,j}^{(l)}} = \frac{1}{M} \sum_{m=1}^{M} \delta_k^{(l)}(m) y_j^{(l-1)}(m), \quad 1 \le k \le n_l, \ 0 \le j \le n_{l-1}, \ 1 \le l \le L. \tag{2.16}$$

An initial guess $\mathbf{w}_0$ for the weight vector $\mathbf{w}$ is set, and the weights are iteratively updated: $\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + \mathbf{d}^{\text{new}}$ (where the vector $\mathbf{d}^{\text{new}}$ can have different expressions, depending on the method being used) until $E(\mathbf{w})$ becomes sufficiently small. At each iteration, a sweep of the training set is performed and $\mathbf{g}^{\text{new}} \stackrel{\text{def}}{=} \nabla E(\mathbf{w}^{\text{old}})$, $\mathbf{H}^{\text{new}} \stackrel{\text{def}}{=} \nabla^2 E(\mathbf{w}^{\text{old}})$ can be computed. When using the steepest descent algorithm we define $\mathbf{d}^{\text{new}} \stackrel{\text{def}}{=} -\alpha_{\text{w}}^{\text{new}} \mathbf{g}^{\text{new}}$, where $\alpha_{\text{w}}^{\text{new}}$ is the current step size, or the *learning rate* (which can be kept constant or can be adjusted at each iteration). In Newton's method, $\mathbf{d}^{\text{new}} \stackrel{\text{def}}{=} -(\mathbf{H}^{\text{new}})^{-1} \mathbf{g}^{\text{new}}$. In order to accelerate convergence, the so-called steepest descent method *with momentum* [86, 190, 229] can be used, where $\mathbf{d}^{\text{new}} \stackrel{\text{def}}{=} \gamma^{\text{new}} \mathbf{d}^{\text{old}} - (1 - \gamma^{\text{new}}) \alpha_{\text{w}}^{\text{new}} \mathbf{g}^{\text{new}}$, with $0 < \gamma^{\text{new}} < 1$. However, there is no clear way of choosing the value of γ^{new}.

When using the conjugate gradient (CG) algorithm, $\mathbf{d}^{\text{new}} \stackrel{\text{def}}{=} \alpha_{\text{w}}^{\text{new}} \mathbf{p}^{\text{new}}$, where $\mathbf{p}^{\text{new}} \stackrel{\text{def}}{=} -\mathbf{g}^{\text{new}}$ at the beginning of training. Subsequently, at each iteration: $\mathbf{p}^{\text{new}} \stackrel{\text{def}}{=} -\mathbf{g}^{\text{new}} + \beta^{\text{new}} \mathbf{p}^{\text{old}}$, where β^{new} can be chosen in different ways [86, 193]:

$$\beta^{\text{new}} = \frac{||\mathbf{g}^{\text{new}}||_2^2 - \mathbf{g}^{\text{new}} \cdot \mathbf{g}^{\text{old}}}{||\mathbf{g}^{\text{old}}||_2^2} \quad \text{(Polak-Ribière)}, \ \beta^{\text{new}} = \frac{||\mathbf{g}^{\text{new}}||_2^2}{||\mathbf{g}^{\text{old}}||_2^2} \quad \text{(Fletcher-Reeves)}. \tag{2.17}$$

The search direction has to be periodically reset to $\mathbf{p}^{\text{new}} = -\mathbf{g}^{\text{new}}$ in order for the conjugate gradient method to converge when minimizing a non-quadratic functional [86].

Different variations of Newton's method are used by different authors [131, 214]. A widely used modified Newton-type method for error functions $E(m) = ||\mathbf{e}(\mathbf{w})||_2^2$ (where the vector $\mathbf{e}(\mathbf{w}) \stackrel{\text{def}}{=} [e(1, \mathbf{w}), e(2, \mathbf{w}), \ldots, e(M, \mathbf{w})]^T \in \mathbf{R}^M$ is the output error vector for the M training patterns) is the *Levenberg-Marquardt (LM)* algorithm [86, 87]. If $I_{n_{\text{w}}}$ is the n_{w}-dimensional identity matrix, and $\mathbf{J}(\mathbf{w})$ is the $M \times n_{\mathrm{w}}$-type matrix with elements $\{\partial e(m,\mathbf{w})/\partial w;\ 1 \le m \le M,\ w \text{ component of } \mathbf{w}\}$, then the LM algorithm defines

$$\mathbf{d}^{\mathrm{new}} \overset{\mathrm{def}}{=} [\mathbf{J}^T(\mathbf{w}^{\mathrm{old}})\mathbf{J}(\mathbf{w}^{\mathrm{old}}) + \mu^{\mathrm{new}} I_{n_{\mathrm{w}}}]^{-1}\mathbf{J}^T(\mathbf{w}^{\mathrm{old}})\mathbf{e}(\mathbf{w}^{\mathrm{old}}), \tag{2.18}$$

where μ^{new} is an adjustable parameter. The main drawback of the Newton-type algorithms is that they involve the calculation of the Hessian matrix and of the solution of a linear system, operations which are computationally expensive. Training methods based on the use of the Extended Kalman Filter (EKF) have also been proposed [45, 217, 223], but they are beyond the scope of this study.

A well-known disadvantage of the point-by-point optimization algorithms is that in general the vector $\mathbf{w}$ converges to a local minimum $\mathbf{w}_0^*$ of $E(\mathbf{w})$ that is close to the initial guess $\mathbf{w}_0$. If $E(\mathbf{w}_0^*)$ is small enough for our purposes then $\mathbf{w}_0^*$ may be an acceptable weight vector for our ANN. However, in general, $E(\mathbf{w}_0^*)$ may not be small enough, and the training has to be restarted with a different initial guess. The practical difficulty is that there is no clear way of setting a good initial guess $\mathbf{w}_0$. Traditionally, the ANN weights are initialized with small random values, but other options may also be considered. A few authors have started to investigate consistent ways of choosing the initial guess for ANN training [99, 256]. However, this type of research is still in the infant stage. In our study, we propose a method for weight initialization for our particular class of problems, that greatly improves the accuracy and robustness of the ANN.

In order to overcome the problem of converging to local minima, global optimization methods, such as the genetic algorithm [80, 130, 161, 168, 177, 221], or the simulated annealing algorithm [42, 73, 235], have been proposed. These algorithms perform a search of the entire input space rather than a local search (around an initial guess) like the classical algorithms. They are guaranteed to 'eventually' converge to the global minimum of the function to be optimized. However, due to the 'curse of dimensionality', these methods require an astronomically long time to converge if the dimension of the search space is large. Most applications of these methods in the literature are reported for dimensions of the input space of the order of tens, while in our experiments the search space has a dimension (which is, actually, the total number of ANN weights) of the order of hundreds or even thousands. Therefore the use of these methods is impractical in our study.

An important question that arises in practice is when to terminate the ANN training. The network must learn the training set $\{\tilde{\mathbf{x}}(1),\tilde{\mathbf{y}}(1)\}$, $\{\tilde{\mathbf{x}}(2),\tilde{\mathbf{y}}(2)\}$, …, $\{\tilde{\mathbf{x}}(M),\tilde{\mathbf{y}}(M)\}$ up to a level that would allow it to correctly generalize, i.e., to provide a correct output $\mathbf{y}$ to a previously "unseen" input $\mathbf{x}$ from the same domain as the input patterns. One can regard the training set as a set of samples of a continuous function $\tilde{\mathbf{\Phi}}$ on a certain region $D \in \mathbf{R}^{n_0}$. That is, $\tilde{\mathbf{y}}(m) = \tilde{\mathbf{\Phi}}(\tilde{\mathbf{x}}(m))$, $1 \le m \le M$. The function $\tilde{\mathbf{\Phi}}$ represents the desired input-output relationship that would make it possible for the ANN to correctly perform a specific task. In most practical situations, there is a lower bound θ for the desired level of accuracy of the training set learning.

At this level of accuracy, the ANN has extracted all the information about the underlying pattern $\tilde{\mathbf{\Phi}}$ that generated them. If the network training is continued beyond this point, then, the lower the training error becomes (i.e., the more accurately the training patterns are learned), the poorer the ANN's generalization performance becomes. The reason is that the network starts to learn features of the particular samples $\{\tilde{\mathbf{x}}(1), \tilde{\mathbf{y}}(1)\}$, $\{\tilde{\mathbf{x}}(2), \tilde{\mathbf{y}}(2)\}$, …, $\{\tilde{\mathbf{x}}(M), \tilde{\mathbf{y}}(M)\}$ taken from the function $\tilde{\mathbf{\Phi}}$, that are not relevant to the general pattern $\tilde{\mathbf{\Phi}}$ anymore. This phenomenon, known as *overfitting*, occurs often in ANN practice [4, 86, 117]. The problem of avoiding overfitting is still an open problem.

The overfitting phenomenon suggests that there is a tradeoff in ANN training between how well the network learns the training patterns and how accurately it can generalize to the unseen inputs. Several methods have been proposed to improve the generalization performance of ANNs. A widely used method is to add noise in the input and/or output training prototypes [23, 199, 247] or in the weight parameters [171]. Another method, the so-called *target smoothing*, involves convoluting the target function with a noise probability density function [199]. A widely used approach is to define the performance index as a sum of two functionals

$$E(\mathbf{w}) = E_{\text{err}}(\mathbf{w}) + \lambda E_{\text{reg}}(\mathbf{w}), \quad \lambda > 0 \tag{2.19}$$

where $E_{\text{err}}(\mathbf{w})$ is a measure of the training error (usually the mean squared error) and $E_{\text{reg}}(\mathbf{w})$ is a regularization term that is designed to improve the generalization performance of the ANN [61]. The *weight decay* approach to generalization improvement involves choosing $E_{\text{reg}}(\mathbf{w}) \overset{\text{def}}{=} ||\mathbf{w}||_2^2$ [142]. In the *error regularization approach*, we define $E_{\text{reg}}(\mathbf{w}) \overset{\text{def}}{=} ||\nabla_{\mathbf{x}}\ \mathbf{y}(\mathbf{x}, \mathbf{w})||_2^2$ [199]. The *weight smoothing* approach reported in [106] is given by

$$E_{\text{reg}}(\mathbf{w}) \overset{\text{def}}{=} \sum_{k=1}^{n_1} \sum_{j=2}^{n_0} \left[w_{k,j}^{(1)} - w_{k,j-1}^{(1)} \right]^2 \tag{2.20}$$

for a 2LFFNN. Bishop [20] proposed using (for an L-layer FFNN) the *curvature-driven smoothing*, where

$$E_{\text{reg}}(\mathbf{w}) \overset{\text{def}}{=} \sum_{m=1}^{M} \sum_{k=1}^{n_L} \sum_{j=2}^{n_0} \left[\frac{\partial^2 y_k^{(L)}}{\partial x_j^2} (\mathbf{x}(m), \mathbf{w}) \right]^2 . \tag{2.21}$$

Drucker and Cun [60] proposed a method called the *double backpropagation*, in which:

$$E_{\text{reg}}(\mathbf{w}) \overset{\text{def}}{=} \sum_{j=1}^{n_0} \left[\sum_{m=1}^{M} \sum_{k=1}^{n_L} e_k(m, \mathbf{w}) \frac{\partial y_k^{(L)}}{\partial x_j} (\mathbf{x}(m), \mathbf{w}) \right]^2 . \tag{2.22}$$

Note that the parameter λ in (2.19) reflects the relative weight of the two terms in the total error function. The practical difficulty when using such a method is that there is no clear way of choosing an acceptable value of λ in a given application. If λ is too small, the regularization term will have no effect, while if λ is too large, all weights may converge to zero

[64]. Basically, penalty terms tend to create additional local minima in the error surface, which makes training even more difficult. In our experiments, we do not use any regularization term in the error function. However, the idea from (2.20), that smoothing the ANN weights improves generalization, is applied in a different fashion, namely by explicitly scaling the weights in the ANN output function rather than in the error function, which makes it unnecessary to choose an appropriate value for a parameter such as λ from (2.19).

A disadvantage of using gradient-type optimization algorithms based on the error backpropagation is that the ANN training usually takes a long time. In order to accelerate convergence, various improvements to the classical optimization approach in ANN training have been proposed [114], such as dynamical adaptation of the learning rate, dynamic adaptation of the weight adjustments (e.g., a momentum term, weight decay), etc. The weight extrapolation approach [114] provides, at each training iteration, estimates for the values of all ANN weights, that are further refined by the current weight update. Layer-by-layer optimization methods [66, 262] combine gradient computation with solving linear systems in order to determine appropriate values for the ANN weights.

2.3 The Universal Approximation Property of ANNs

Applications on approximating a nonlinear function $\tilde{f} : \mathbf{R}^{n_0} \to \mathbf{R}$ by the output of a MLFFNN have been widely investigated during the last 15 years. More specifically, for every $\mathbf{x}$ in a subset of $\mathbf{R}^{n_0}$, the value $\tilde{f}(\mathbf{x})$ is approximated by the output $y(\mathbf{x}, \mathbf{w})$ of a certain MLFFNN with a single neuron in the output layer (SMLFFNN, for brevity). Various authors have tried to answer the following questions. What classes of functions can be approximated by the output of a SMLFFNN ? What transfer functions should be used in each layer ? How many neurons are required in each of the hidden layers ?

In most cases, two-layer feed-forward (2LFF) networks with a single output neuron (S2LFFNNs), with sigmoidal transfer function in the hidden layer and linear transfer function in the output layer (S2LFF1S2LNNs) are considered. In general, if x_1, x_2, …, x_{n_0} are the inputs of a 2LFFNN with n_1 neurons in the first (hidden) layer and n_2 neurons in the second (output) layer, then the ANN output has the expression

$$y_i^{(2)} = f^{(2)} \left\{ w_{i,0}^{(2)} + \sum_{k=1}^{n_1} w_{i,k}^{(2)} f^{(1)} \left(w_{k,0}^{(1)} + \sum_{h=1}^{n_0} w_{k,h}^{(1)} x_h \right) \right\}, \quad 1 \le i \le n_2. \tag{2.23}$$

Let $\mathcal{N}[n_0, n_1, n_2; f^{(1)}, f^{(2)}]$ be the class of functions $N[n_0, n_1, n_2; f^{(1)}, f^{(2)}; \mathbf{w}] : \mathbf{R}^{n_0} \to \mathbf{R}^{n_2}$ defined by (2.23), that express the output $\mathbf{y}^{(2)}$ of a 2LFFNN in terms of its inputs. Note that

$$\mathbf{w} \overset{\text{def}}{=} [w_{1,0}^{(1)}, \ldots w_{1,n_0}^{(1)}, w_{2,0}^{(1)}, \ldots w_{n_1,n_0}^{(1)}, w_{1,0}^{(2)}, \ldots w_{1,n_1}^{(2)}, w_{2,0}^{(2)}, \ldots w_{n_2,n_1}^{(2)}]^T \tag{2.24}$$

is the vector of all ANN weights ($[\cdot]^T$ is the transpose operator). Denote by $\mathcal{N}[n_0, *, n_2; f^{(1)}, f^{(2)}]$ the set of all functions implemented by such an ANN with an arbitrarily large number of hidden neurons [96]:

$$\mathcal{N}[n_0, *, n_2; f^{(1)}, f^{(2)}] = \bigcup_{n_1=1}^{\infty} \mathcal{N}[n_0, n_1, n_2; f^{(1)}, f^{(2)}] \tag{2.25}$$

A function $\sigma(\cdot)$ is called a (generalized) *sigmoidal* function (or a *squashing function*) if

$$-\infty < \lim_{x\to-\infty} \sigma(x) < \lim_{x\to+\infty} \sigma(x) < +\infty. \tag{2.26}$$

Note that $\sigma(\cdot)$ is not necessarily continuous or monotone. Examples of sigmoidal functions are [34, 77]: $\sigma(x) = \tanh(x)$ (the *hyperbolic tangent squasher*), $\sigma(x) = H(x)$ (Heaviside's *unit step function*), $\sigma(x) = \operatorname{lsig}(x) = 1/[1 + \exp(-x)] = [1 + \tanh(x)]/2$ (the *logistic sigmoid*),

$$\sigma(x) = \begin{cases} 0, & x \le -\pi/2 \\ \dfrac{1 + \cos(x + 3\pi/2)}{2}, & -\pi/2 \le x \le \pi/2 \\ 1, & x \ge \pi/2 \end{cases} \quad \text{(the } \textit{cosine squasher} \text{ [77]).} \tag{2.27}$$

Most approximation results state that, under certain assumptions on the function $\sigma(\cdot)$, the set $\mathcal{N}[n_0, *, 1; \sigma, I]$ ($I(x) = x, \forall x$) is dense in some metric space $(\mathcal{F}, \rho)$ of real-valued functions defined on some subset of $\mathbf{R}^{n_0}$. In most cases $\mathcal{F}$ is a space of continuous or L^p-integrable functions, and the metric ρ is usually (but not always) generated by some norm $||\cdot||$ on $\mathcal{F}$, namely $\rho(f,g) = ||f - g||$, $\forall f, g \in \mathcal{F}$. The density of $\mathcal{N}[n_0, *, 1; \sigma, I]$ in $(\mathcal{F}, \rho)$ is equivalent to saying that any function in $\mathcal{F}$ can be approximated (in the metric ρ) with any desired accuracy by a function in $\mathcal{N}[n_0, *, 1; \sigma, I]$. In other words, any function in $\mathcal{F}$ can be approximated (in the metric ρ) with any desired accuracy by a function given by the output of a 2LFF1S2LNN, provided that sufficiently many neurons in the hidden layer are available:

$$\forall f \in \mathcal{F},\ \forall \epsilon > 0\ \exists n_1(f,\epsilon),\ \exists \mathbf{w}(f,\epsilon):\ \rho\left(\mathcal{N}[n_0, n_1(f,\epsilon), 1; \sigma, I; \mathbf{w}(f,\epsilon)], f\right) < \epsilon. \tag{2.28}$$

Various approximation results have been proved by Cybenko (1989) [53], Hornik, Stinchcombe, and White (1989) [97], Funahashi (1989) [75], Ito (1991) [105], Hornik (1991) [96], Gallant and White (1992,1998) [77, 76], etc. Chen, Chen, and Liu (1993) generalized the classical approximation results [31, 32, 33, 34]. They showed that, if σ is a bounded sigmoid, then $\mathcal{N}[n_0, *, 1; \sigma, I]$ is dense in the space

$$C(\bar{\mathbf{R}}^{n_0}) \stackrel{\text{def}}{=} \{f \in C(\mathbf{R}^{n_0}); \exists \lim_{||\mathbf{x}||_2\to\infty} f(\mathbf{x}) < \infty\} \tag{2.29}$$

with respect to the $|||\cdot|||_C$ norm. For $n_0 = 1$, Chen, Chen, and Liu proved that $\mathcal{N}[1, *, 1; \sigma, I]$ is dense in the space

$$\{f \in C(\mathbf{R}^1);\ \lim_{x\to-\infty} f(x),\ \lim_{x\to+\infty} f(x) \text{ exist and are finite}\} \supset C(\bar{\mathbf{R}}^1). \tag{2.30}$$

In particular, $\mathcal{N}[n_0, *, 1; \sigma, I]$ is dense in $C(K)$, for any compact $K \subset \mathbf{R}^{n_0}$. Also, if $\sigma(\cdot)$ is a sigmoid, $\sigma \in L^p_{\text{loc}}(\mathbf{R})$, then $\mathcal{N}[n_0, *, 1; \sigma, I]$ is dense in $L^p(K)$, for any compact $K \subset \mathbf{R}^{n_0}$, with respect to the L^p norm

$$(2.31) \qquad |||f|||_p \stackrel{\text{def}}{=} \left(\int_K |f(\mathbf{x})|^p \right)^{1/p}, \ 1 \le p < \infty, \ |||f|||_\infty \stackrel{\text{def}}{=} \operatorname{ess\,sup} \{ |f(\mathbf{x})|; \mathbf{x} \in K \} .$$

Most of the classical approximation results were proven assuming that the sigmoid $\sigma(\cdot)$ is either continuous (Cybenko [53]) or non-decreasing (Hornik, Stinchcombe, and White [97]), or a particular class of sigmoids was used (the cosine squasher, Gallant and White [77]). Ito [105] proved that, for some particular types of sigmoids, including the Heaviside unit step fucntion, $\mathcal{N}[n_0, *, 1; \sigma, I]$ is dense with respect to the $||| \cdot |||_C$ norm in the space of rapidly decreasing continuous functions:

$$(2.32) \qquad \left\{ f \in C(\mathbf{R}^{n_0}); \lim_{||\mathbf{x}|| \to \infty} |x_1^{k_1} x_2^{k_2} \dots x_{n_0}^{k_{n_0}} f(\mathbf{x})| < \infty, \forall k_1, \dots k_{n_0} \in \mathbf{Z}_+, \forall \mathbf{x} \in \mathbf{R}^{n_0} \right\} .$$

Since in the present work a 2LFF1SNN with first-layer transfer function $\sigma(x) = \tanh(x)$ is used in all experiments, the approximation theorem relevant to this study is Cybenko's result [53]:

Theorem 2.3.1. If $K \subset \mathbf{R}^{n_0}$ is a compact set and σ is a continuous sigmoid, then $\mathcal{N}[n_0, *, 1; \sigma, I]$ is dense in $C(K)$:

$$(2.33) \ \forall f \in C(K), \ \forall \epsilon > 0 \ \exists n_1(f, \epsilon), \ \exists \mathbf{w}(f, \epsilon) : \ |||N[n_0, n_1(f, \epsilon), 1; \sigma, I; \mathbf{w}(f, \epsilon)] - f|||_C < \epsilon \ [53].$$

In other words, any continuous function on a compact K in $\mathbf{R}^{n_0}$ can be uniformly approximated to any desired degree of accuracy by the mapping that provides the output of a 2LFF1S2LNN in terms of the ANN inputs, if sufficiently many neurons are used in the first (hidden) layer. In practice, the function $f \in C(K)$ is interpolated by the ANN from a finite set of known values of f in points of K (which constitute the training set for the neural network). The network weights $\mathbf{w}(f, \epsilon)$ are determined by training the ANN based on the set of known values of f. At the end of training, the ANN should be capable of interpolating the function f on K with accuracy at most ϵ.

Funahashi [75] proved that, if σ is a function that is nonconstant, bounded, increasing, and continuous, then $\forall L \ge 2$, $\forall K$ compact in $\mathbf{R}^{n_0}$, any function in $C(K)$ can be uniformly approximated with any accuracy by the output of a L-layer FFNN with a linear transfer function in the output layer and with σ as the transfer function in all hidden layers. It is possible that in some cases a L-layer ANN might perform a good approximation with less computational cost, that is, using less neurons, and thus less ANN weights, than a 2LFFNN.

Denote by $\mathcal{B}^{n_0}$ the Borel field of $\mathbf{R}^{n_0}$, and by $\mathcal{M}^{n_0}$ the set of all Borel measurable functions from $\mathbf{R}^{n_0}$ to $\mathbf{R}$. Hornik, Stinchcombe, and White [97] proved that, if σ is a non-decreasing sigmoid and μ is any probability (or finite) measure on $(\mathbf{R}^{n_0}, \mathcal{B}^{n_0})$, then $\mathcal{N}[n_0, *, 1; \sigma, I]$ is dense in $\mathcal{M}^{n_0}$ with respect to the metric

$$(2.34) \qquad \rho_\mu(f,g) \overset{\text{def}}{=} \inf\left\{\epsilon > 0; \mu\{\mathbf{x} : |f(\mathbf{x}) - g(\mathbf{x})| > \epsilon\} < \epsilon\right\}, \quad \forall f,g \in \mathcal{M}^{n_0}.$$

Note that $\rho_\mu(f,g)$ is small if and only if there is a small probability that f and g differ significantly, and $\rho_\mu(f,g) = 0$ if and only if f and g are μ-equivalent, i.e., $\mu\{\mathbf{x} : f(\mathbf{x}) = g(\mathbf{x})\} = 1$.

Several authors have investigated the approximation of the derivatives of a mapping by 2LFF ANNs. Hornik [96] proved that if $\sigma \in C^m(\mathbf{R})$ has all its derivatives up to order m bounded, then $\forall K$ compact in $\mathbf{R}^{n_0}$, any function in $C^m(K)$ can be approximated in any L^p norm, $1 \leq p \leq \infty$, together with all its derivatives up to order m, with any accuracy, by some function in $\mathcal{N}[n_0, *, 1; \sigma, I]$. Gallant and White [76] showed that, when a 2LFFNN is trained by a least-square minimization to approximate an unknown mapping, then it will automatically approximate the derivatives of that mapping as well (up to a certain degree m, depending on the regularity of σ). It is remarkable that we do not have to explicitly provide the values of those derivatives as targets in the ANN training.

Castro, Mantas, and Benítez [27] proved that, if σ is a non-decreasing sigmoid and ϕ is a strictly increasing, continuous sigmoid, then $\mathcal{N}[n_0, *, n_2; \sigma, \phi]$ is dense in the set of Borel measurable functions defined on some compact $K \in \mathbf{R}^{n_0}$ with values in $[R(\phi)]^{n_2}$, where $R(\phi)$ is the range of ϕ (which is an interval, or the whole $\mathbf{R}$). Hence, the universal approximation property of ANNs holds even if the transfer function of the second layer is a sigmoid.

As pointed out by Hornik, Stinchcombe, and White [97], the functions encountered in practical applications are in general Borel measurable, therefore the approximation results for Borel measurable functions are sufficient to justify the use of ANNs in practical problems. Hornik [96] observed that, since the conditions that need to be imposed on σ in order to achieve universal approximation are quite weak, it must be the MLFF architecture itself rather than the choice of the transfer functions that provides the ANNs with the universal approximation property. He also notes that, even though in principle one can use a large class of transfer functions, not all functions will perform equally well in specific practical problems.

Basically, the above results state that a nonlinear function can be approximated to any desired accuracy by the output of a 2LFFNN provided that sufficiently many neurons are available in the network's hidden layer. However, Cybenko [53] notes that the number of neurons necessary in most practical problems is likely astronomically high, due to the so-called *curse of dimensionality*. In a practical approximation problem, it is also necessary that there are sufficiently many training examples for the ANN, and that they are well distributed in the domain $D \in \mathbf{R}^{n_0}$ on which the unknown function is defined. Unfortunately, we may not have control over how well the training examples are distributed inside D in some applications. The best distribution of the interpolation points is closely related to the curvature of the function to be approximated, which is unknown. The next best solution is to have a lot of interpolation points uniformly distributed in D. Note that, for the experiments performed in the present study, the training data set is not very large (containing only a few oscillation cycles), therefore we cannot count on this situation to occur either. In our study, we find a means for effectively extracting information from the training set for a given number of neurons. Hence, for a given application, fewer neurons will be necessary. As for the distribution of training examples, we propose a criterion to assess, for our class of applications, whether the training set is appropriate or not. These issues will be discussed in detail in the following chapters.

In general, how to choose the number of ANN inputs and the number of hidden layer neurons for a given application is still an unsolved problem. Having too few parameters (weights) in the network leads to poor generalization because the ANN system is less complex than the unknown mapping $\tilde{\mathbf{\Phi}}$, hence the mapping cannot be learned. Too many parameters implies that the ANN system is more complex than the unknown mapping $\tilde{\mathbf{\Phi}}$. Consequently, the mapping $\tilde{\mathbf{\Phi}}$ may not be uniquely determined by its samples because the ANN has too many degrees of freedom.

Different types of constructive and destructive algorithms have been designed to optimize the complexity of a neural network. Constructive algorithms [26, 40, 79, 104, 191, 214] iteratively increment the number of neurons in a certain layer. The ANN is first trained with a single neuron in the respective layer. If the performance index is not small enough, then a neuron is added to the working layer and its weights are trained while the other ANN weights are kept frozen. The process is repeated until the value of the performance index, or of other relevant quantities (such as, for instance, different types of information criteria) is small enough. Destructive algorithms (or *pruning* algorithms) [51, 64, 223, 227] start by training a network with a lot of neurons, and then different neurons or weights are deleted based on their 'importance', or 'relevance' for the network (given by a certain 'relevance' index, defined in most cases based on some kind of statistical reasoning). We have not experimented with these methods in the present study, but they could constitute interesting subjects for further research.

Chapter 3

Main Contribution

Suppose we know a set of consecutive terms of a time series $s(1)$, $s(2)$, …, $s(t)$ …, $s(t_1)$ (where t are discrete moments of time). These could represent, for instance, measurements at equal time intervals taken from a process that is continuous in time. We wish to predict $s(t_1+1)$, $s(t_1+2)$, …, $s(t_2)$, for t_2 much larger than t_1. In this study, we assume that the time series arises from a model representing a nonlinear dynamical system. Moreover, the system is self-excited, which implies that no external forcing is imposed after $t = t_1$.

3.1 Time Series Models

In order to define the context of our main contribution in this study, a brief review of the time series models developed up to this moment will be performed in this section. An excellent review of time series analysis methods has been performed by Fan and Yao (2003) [68]. In time series analysis, the discrete-time trajectory $\{s(t)\}_t$ is viewed as the realization of a stochastic process $\{X_t\}_t$, and the goal is to determine the probability law that governs a segment of the observed data $s(t_0+1)$, $s(t_0+2)$, …, $s(t_1)$ (for some $t_0 < t_1$), in order to forecast the unknown observations $s(t_1+1)$, $s(t_1+2)$, … [68, 215]. Since the number of known observations is always finite, there are infinitely many stochastic processes that can generate the same observed data set [68]. Therefore the usual approach is to search for the probability law in a specified family (the *modelling* stage), and then to select a member in that family that best fits the known data set (the *estimation* stage). When the probability law is specified up to a finite number of parameters to be determined, we are dealing with a *parametric model*. When there are an infinite number of parameters or if the form of the probability law is not completely specified, we are dealing with a *nonparametric model* [68]. A detailed presentation of time series analysis methods is beyond the scope of this study. We will only point out some issues that are relevant to our approach. The application of some classical time series models to the prediction problem tackled in this study is reported in detail in [194, 258, 259, 260].

A stochastic process $\{\epsilon_t\}_t$ for which $\forall t$: $\mathrm{E}(\epsilon_t) = 0$, $\mathrm{Var}(\epsilon_t) = \sigma^2 < \infty$, and $\mathrm{Cov}(\epsilon_t, \epsilon_s) = 0$ for $s \neq t$, is called *white noise* (denote $\{\epsilon_t\}_t \sim \mathrm{WN}(0, \sigma^2)$). A special case of a white noise process is a sequence $\{\epsilon_t\}_t$ of independent and identically distributed random variables with $\mathrm{E}(\epsilon_t) = 0$ and $\mathrm{Var}(\epsilon_t) = \sigma^2 < \infty$ $\forall t$. We denote such a stochastic process by $\{\epsilon_t\}_t \sim \mathrm{IID}(0, \sigma^2)$ [68].

The most widely used parametric model in time series analysis is the so-called *autoregressive moving average (ARMA)* model

$$X_t = \sum_{h=1}^{p} a_h X_{t-h} + \epsilon_t + \sum_{j=1}^{q} b_j \epsilon_{t-j} \tag{3.1}$$

where $\{\epsilon_t\}_t \sim \mathrm{WN}(0, \sigma^2)$ and a_h, b_j are parameters to be estimated by statistical methods based on the known observations [68].. Once the parameters have been determined, a prediction for the observation at the next time moment can be generated [68, 215]. If $a_h = 0$ $\forall h$ in (3.1), we obtain a *moving average (MA)* model, while if $b_j = 0$ $\forall j$, we are dealing with an *autoregressive (AR)* model. Note that such models are representative for the *time domain approach*, which describes the structure of a time series in terms of the dependence of the current observation on the past observations. There is also a *frequency domain approach*, which describes the underlying dynamics in terms of the systematic sinusoidal variations present in the data [215], and which has not been investigated in our study.

Sometimes it is necessary to differentiate the time series (that is, to replace the original data set by the sequence of the differences between each observation and the previous one), in order to remove some time trends (such as a steady increase in mean, for instance). When an ARMA model is fit to the *d* times differentiated data set, the predictions for the subsequent observations of the original time series can be generated by integrating *d* times the predicted values for the differentiated series. This is referred to as an *autoregressive integrated moving average (ARIMA)* process [68].

Despite its excellent theoretical tractability, the main shortcoming of the ARIMA model is its linear nature, which makes it unable to model time series with underlying nonlinear structure. Nonlinear time series models have been developed for that purpose. However, this is still a poorly developed research field with most of the work done since the 1980's. In this section we only point out some issues relevant to our research topic.

A popular nonlinear time series model, which is introduced to capture the volatility of financial time series, is the *generalized autoregressive conditional heteroscedastic (GARCH)* model:

$$X_t = \sigma_t \epsilon_t, \quad \sigma_t^2 = a_0 + \sum_{h=1}^{p} a_h X_{t-h}^2 + \sum_{j=1}^{q} b_j \sigma_{t-j}^2 \tag{3.2}$$

where $\{\epsilon_t\}_t \sim \mathrm{IID}(0, 1)$, and $a_h \geq 0$, $b_j \geq 0$ [68]. If $b_j = 0$ $\forall j$, we obtain the *autoregressive conditional heteroscedastic (ARCH)* model. The *threshold autoregressive (TAR)* model performs a piecewise linear approximation of the data, the division of the state space being dependent on a threshold variable X_{t-d} (for some $d \geq 1$):

$$X_t = \sum_{i=1}^{k} \left\{ a_{i,0} + \sum_{h=1}^{p_i} a_{i,h} X_{t-h} \right\} I_{A_i}(X_{t-d}) + \sigma_i \epsilon_t \tag{3.3}$$

where $I_{A_i}(\cdot)$ is the characteristic function of the set A_i, $\{\epsilon_t\}_t \sim \mathrm{IID}(0,1)$, $\sigma_i > 0$, and $\{A_i\}_i$ forms a partition of $\mathbf{R}$ [68]. The TAR model is not as well understood as the ARIMA model. A *bilinear model* has the form:

$$X_t = \sum_{h=1}^{p} a_h X_{t-h} + \epsilon_t + \sum_{j=1}^{q} b_j \epsilon_{t-j} + \sum_{h=1}^{P} \sum_{j=1}^{Q} c_{h,j} X_{t-h} \epsilon_{t-j} \tag{3.4}$$

where $\{\epsilon_t\}_t \sim \mathrm{IID}(0, \sigma^2)$. This model is capable of capturing occasional outbursts in time series, which might be useful for modelling seismological data [68].

Even though parametric models are powerful tools when the models are correctly specified, the main difficulty lies in the choice of the specified model for a given time series. In order to improve the accuracy of the approximation, models of a more general form need to be constructed. One of the most general models is the *saturated (full) nonparametric model*:

$$X_t = f(X_{t-1}, \ldots, X_{t-p}) + \sigma(X_{t-1}, \ldots, X_{t-p})\epsilon_t \tag{3.5}$$

where $\{\epsilon_t\}_t \sim \mathrm{IID}(0,1)$ and ϵ_t is independent of X_{t-h}, $h \geq 1$ [68]. The function $f(\cdot)$ is the *autoregressive function*, while $\sigma(\cdot)$ is the *conditional variance function*. Though modelling biases are reduced by using this type of model, the estimation of the autoregressive function (by methods such as multivariate local polynomial regression, spline interpolation, etc [68]) is not feasible due to the 'curse of dimensionality'. Therefore nonparametric models with particular forms of the autoregressive function have been developed [68].

The *functional coefficient model (FAR)* has the form:

$$X_t = \sum_{h=1}^{p} f_h(X_{t-d}) X_{t-h} + \sigma(X_{t-d})\epsilon_t \tag{3.6}$$

for some $d \geq 1$, where $\{\epsilon_t\}_t \sim \mathrm{IID}(0,1)$ and ϵ_t is independent of X_{t-h}, $h \geq 1$, $\forall t$. If, instead of X_{t-d}, each $f_h(\cdot)$ takes as argument a linear combination of past observations, then we obtain the *adaptive functional coefficient autoregresssive model* [68]. A parametric version of the FAR model is given by the *exponential autoregressive (EXPAR)* model:

$$X_t = \sum_{h=1}^{p} \{\alpha_h + (\beta_h + \gamma_h X_{t-d}) \exp(-\theta_i X_{t-d}^2)\} X_{t-h} + \epsilon_t \tag{3.7}$$

proposed by Ozaki, where $\theta_i \geq 0$ [68]. Ozaki showed that the EXPAR model is capable of capturing complex nonlinear phenomena such as amplitude-dependent frequency, jump phenomena and limit cycles. The model was subsequently extended by replacing the coefficients γ_h by Hermite-type polynomials in X_{t-d} [88, 89, 182, 183]. This particular model has been applied to data sets used in our study, and its advantages and disadvantages are reported in [194, 258, 259, 260]. Several other nonparametric models include *additive models*, *two-term interaction models*, etc [68].

3.2 The Proposed Approach

In our approach, we are modelling a segment $s(t_0+1)$, $s(t_0+2)$, ..., $s(t_1)$ (for some $t_0 < t_1$) of the known data set using a nonlinear mapping $\Phi_{\mathbf{w}}$ given by the output of an ANN ($\mathbf{w}$ denotes the vector of all ANN weights). It has been proven that, for all practical purposes, a nonlinear function can be interpolated from examples of known values, with any desired accuracy, by the mapping that gives the output of a two-layer feed-forward ANN with a sigmoidal transfer function in the first layer (a 2LFF1SNN) and a linear transfer function in the second layer (a 2LFF1S2LNN), provided that sufficiently many neurons are available in the network's hidden layer (see Theorem 2.3.1) [31, 32, 33, 34, 53, 75, 76, 77, 96, 97, 105]. A *sigmoidal function* is a bounded function with horizontal asymptotes at both $-\infty$ and $+\infty$, with the left-hand asymptote being lower than the right-hand asymptote. In this study, we use $f^{(1)}(x) = \sigma(x) = \tanh(x)$ as the transfer function in the first layer. The universal approximation property still holds for 2LFF1SNNs with continuous and monotone sigmoidal transfer function in the output layer (2LFF1S2SNNs) [27].

We are implicitly assuming that, for n_0 sufficiently large, there exists a mapping $\bar{\Phi} = \bar{\Phi}_{n_0}$ that models sufficiently well the entire trajectory, i.e.,

$$s(t) = \bar{\Phi}(s(t-1), \ldots, s(t-n_0)) + \bar{e}(t), \quad \forall t \geq n_0 + 1 \tag{3.8}$$

with the modelling error

$$\bar{e}_\infty \overset{\text{def}}{=} \sup\{|\bar{e}(t)|; t \geq n_0 + 1\} \tag{3.9}$$

being sufficiently small. In other words, we are assuming that the dynamics of the given trajectory can be accurately described by the dependence of each current observation on a **finite** number of past observations.

For n_0 sufficiently large, we are attempting to approximate the mapping $\bar{\Phi}$ in (3.8) by a mapping $\Phi_{\mathbf{w}} \in \mathcal{N}[n_0, *, 1; \sigma, f^{(2)}]$ (see equation (2.28)). If the number n_1 of the hidden neurons is sufficiently large, this should be possible in light of the universal approximation property of ANNs. All we need is to have sufficiently many examples of the correct input-output pairs

$$\left\{ \{ \ \mathbf{s}(t-1) \overset{\text{def}}{=} [s(t-1), \ldots, s(t-n_0)]^T, \ s(t) \ \} \ ; \ t_0 + 1 \leq t \leq t_1 \right\} \tag{3.10}$$

for ANN training. The pairs form the *training set* for our application. $\Phi_{\mathbf{w}}$ is the mapping that provides the output $y_{\mathbf{w}}^{(2)}(t)$ of a 2LFF1SNN as a function of its n_0 inputs. The outputs of the

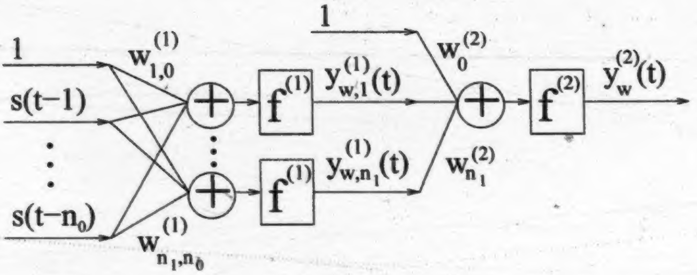

Figure 3.1: Two Layer Feedforward ANN Used for Prediction.

first and second layer of such an ANN (represented in Fig.3.1, with $f^{(1)}(x) = \tanh(x)$) at each time step t, are respectively:

$$y^{(1)}_{\mathbf{w},k_1}(t) = \tanh\left\{ w^{(1)}_{k_1,0} + \sum_{k_0=1}^{n_0} w^{(1)}_{k_1,k_0} s(t-k_0) \right\}, \quad 1 \le k_1 \le n_1 \tag{3.11}$$

$$y^{(2)}_{\mathbf{w}}(t) = f^{(2)}\left\{ w^{(2)}_0 + \sum_{k_1=1}^{n_1} w^{(2)}_{k_1} y^{(1)}_{\mathbf{w},k_1}(t) \right\} \tag{3.12}$$

where $\mathbf{w}$ denotes the vector of all ANN weights $w^{(1)}_{k_1,k_0}$, $w^{(2)}_{k_1}$, $w^{(2)}_0$, $1 \le k_1 \le n_1$, $0 \le k_0 \le n_0$.

An appropriate $\mathbf{w}$ has to be determined such that the errors

$$\tilde{e}_{\mathbf{w}}(t) \overset{\text{def}}{=} s(t) - \Phi_{\mathbf{w}}(s(t-1), \ldots, s(t-n_0)), \quad t_0+1 \le t \le t_1 \tag{3.13}$$

are sufficiently small. The ANN weights for which $\Phi_{\mathbf{w}}$ correctly models the training set are determined by minimizing the mean-squared error (the *performance index*):

$$E(\mathbf{w}) \overset{\text{def}}{=} \frac{1}{t_1 - t_0} \sum_{t=t_0+1}^{t_1} [s(t) - \Phi_{\mathbf{w}}(\mathbf{s}(t-1))]^2 \tag{3.14}$$

by using a nonlinear optimization algorithm. This constitutes the ANN training process. In our study, the conjugate gradient method was applied. At every discrete time step t, the value

$$\tilde{y}_{\mathbf{w}}(t) \overset{\text{def}}{=} \Phi_{\mathbf{w}}(s(t-1), \ldots, s(t-n_0)) \tag{3.15}$$

is called the *one-step prediction (OSP)* for $s(t)$, and the value

$$\tilde{e}_{\mathbf{w}}(t) \overset{\text{def}}{=} s(t) - \tilde{y}_{\mathbf{w}}(t) \tag{3.16}$$

is called the *one-step prediction error (OSE)* at step t.

Once $\mathbf{w}$ has been determined, an estimate for $s(t_1+1)$ will be given by:

$$y_{\mathbf{w}}(t_1+1) \overset{\text{def}}{=} \Phi_{\mathbf{w}}(s(t_1), s(t_1-1), \ldots, s(t_1-n_0+1)) \tag{3.17}$$

If the correct value $s(t_1+1)$ becomes available before the moment $t = t_1 + 2$, the value $s(t_1+2)$ can be estimated by a OSP process in the same manner:

$$s(t_1+2) \approx y_{\mathbf{w}}(t_1+2) \overset{\text{def}}{=} \Phi_{\mathbf{w}}(s(t_1+1), s(t_1), \ldots, s(t_1-n_0+2)) \tag{3.18}$$

Otherwise, the previously predicted value $y_{\mathbf{w}}(t_1+1)$ has to be used in order to predict $s(t_1+2)$:

$$s(t_1+2) \approx y_{\mathbf{w}}(t_1+2) \overset{\text{def}}{=} \Phi_{\mathbf{w}}(y_{\mathbf{w}}(t_1+1), s(t_1), s(t_1-1), \ldots, s(t_1-n_0+2)) \tag{3.19}$$

Similarly, $s(t_1+3)$ will be estimated by:

$$y_{\mathbf{w}}(t_1+3) \overset{\text{def}}{=} \Phi_{\mathbf{w}}(y_{\mathbf{w}}(t_1+2), y_{\mathbf{w}}(t_1+1), s(t_1), \ldots, s(t_1-n_0+3)) \tag{3.20}$$

and so forth. The process described in (3.19) and (3.20) is referred to as a *multi-step (*or *recursive) prediction (MSP)*. Since we are assuming that only a limited transient data set is known and that the subsequent values $s(t_1+1)$, $s(t_1+2)$, ... never become available through measurement during the forecasting process, the OSP formula cannot be used to estimate these unknown values. Instead, at each step $t \geq t_1+1$, the value $s(t)$ will be estimated by its MSP:

$$\hat{y}_{\mathbf{w}}(t) \overset{\text{def}}{=} \Phi_{\mathbf{w}}(\hat{y}_{\mathbf{w}}(t-1), \ldots, \hat{y}_{\mathbf{w}}(t-n_0)) \tag{3.21}$$

where

$$\hat{y}_{\mathbf{w}}(t) \overset{\text{def}}{=} s(t), \quad t_1 - n_0 + 1 \leq t \leq t_1. \tag{3.22}$$

At every step t, the value

$$\hat{e}_{\mathbf{w}}(t) \overset{\text{def}}{=} s(t) - \hat{y}_{\mathbf{w}}(t) \tag{3.23}$$

defines the *multi-step prediction error (MSE)*. Note that one of the main difficulties in performing a MSP process is to control the propagation of the prediction errors. The errors are due to the past predicted values being used in order to generate the current prediction.

The *multi-step-ahead prediction (MSAP)* method has been somewhat investigated in the statistical literature [30, 84, 95, 113, 176, 228]. It involves, at each time step t, predicting $s(t_1+d)$ in terms of the known values $s(t_1)$, $s(t_1-1)$, ..., $s(t_1-n_0+1)$ (for some fixed $d \geq 1$):

$$\tilde{y}_{\mathbf{w}}(t_1+d) \overset{\text{def}}{=} \Phi_{\mathbf{w}}(s(t_1), s(t_1-1), \ldots, s(t_1-n_0+1)). \tag{3.24}$$

For $d = 1$, the MSAP becomes an OSP process. In contrast to this method, what we call MSP is also known as *one-step plug-in method*. It is known in the statistical literature that, for a given $d \geq 1$, a MSAP as in (3.24) provides a more accurate prediction than using d times a MSP (or 'one-step plug-in') process as in (3.21). Note that the target outputs when training the ANN to perform a MSAP as in (3.24) are $s(t_0+d)$, ..., $s(t_1)$. Since sufficient data points are needed for network training, it follows that the lag d has to be much smaller than $t_1 - t_0$ ($d \ll t_1 - t_0$).

In addition, the farthest observation in the future that can be predicted is $s(t_1 + d)$, where $t_1 + d \ll 2t_1 - t_0$. In our applications, our goal is to predict the unknown values $s(t_1 + 1)$, …, $s(t_2)$, where t_2 is considerably larger than t_1. Hence, in order to estimate the value of $s(t_2)$, some kind of 'plug-in' still needs to be performed, at least $t_2 - 2t_1 + t_0$ times (that is, several hundreds of times, in our experiments). In addition, the mapping $\bar{\Phi}^{<d>}$ that models the dependence of $s(t+d)$ on $s(t)$, $s(t-1)$, …, $s(t-n_0+1)$, becomes more complicated as d increases. Hence, while the known data set $s(1)$, …, $s(t_1)$ (where t_1 is of the order of hundreds) might be sufficient for the ANN to accurately approximate the mapping $\bar{\Phi}^{<1>}$, it might not be sufficient to approximate the mapping $\bar{\Phi}^{<10>}$, for instance. Since in our experiments the known data set is quite limited, we used $d = 1$ (and hence a MSP process) in all cases.

Takens [224] proved that for a compact manifold $\mathcal{M}$ of dimension m, a smooth function (an *observable*) $\gamma : \mathcal{M} \to \mathbf{R}$, a dynamical system $\phi : \mathbf{R}_+ \times \mathcal{M} \to \mathcal{M}$, and a point $\mathbf{x}_0 \in \mathcal{M}$, there exists (a countable intersection of open dense sets) $C_{\phi,\mathbf{x}_0} \subset \mathbf{R}_+$ such that, for any $h \in C_{\phi,\mathbf{x}_0}$, there is a smooth embedding of $\mathcal{M}$ into $\mathbf{R}^{2m+1}$ bijectively mapping the ω-limit set $\omega(\mathbf{x}_0) \overset{\text{def}}{=} \{\mathbf{x} \in \mathcal{M}\ ;\ \exists t_n \to \infty \text{ with } \phi_{t_n}(\mathbf{x}_0) \to \mathbf{x}\}$ to the set of limit points of the sequence

$$(3.25) \qquad \left\{\left[\gamma\left(\phi_{(n+0)h}(\mathbf{x}_0)\right), \gamma\left(\phi_{(n+1)h}(\mathbf{x}_0)\right), \ldots, \gamma\left(\phi_{(n+2m)h}(\mathbf{x}_0)\right)\right]^{\mathrm{T}}\right\}_{n=1}^{\infty},$$

which implies (when $\gamma = \pi_1$, the projection on the first component) that we can reconstruct the ω-limit set of an m-dimensional trajectory by using only the information about one of its m components. In some of our experiments, the transient state of only one component of an 8-dimensional aeroelastic system is provided as training set to the neural network. The ANN accurately predicts the limit cycle oscillations on that component even though the 8 first-order nonlinear ODEs are coupled and the other 7 components of the dynamics are not provided [140, 151]. Note that, in order to reconstruct the limit set $\omega(\mathbf{x}_0)$, appropriate values for the time lag h and the embedding dimension m have to be chosen. However, no explicit formulas for computing m and h are available. The information on m and h is usually unavailable for practical problems. Moreover, Takens' theory is applied to a noise-free condition, while for many applications we are interested in data corrupted by noise. The proposed neural network predictor is developed so that it is capable of dealing with noisy data.

In the present thesis we concentrate on predicting a particular class of signals, namely oscillatory signals. More specifically, the signals we consider have the following poperties:

P.1. For every $\tau > 0$ sufficiently large, the mean value $\bar{s}(\tau)$ of $\{s(t); 1 \leq t \leq \tau\}$ is close to zero, and $\tau \mapsto \bar{s}(\tau)$ is an approximately constant mapping. If the mean is not close to zero, then the signal can be rescaled.

P.2. $\{s(t); t \geq 1\}$ is an oscillatory signal, i.e., there are sufficiently many moments $\tilde{t}_m$ such that $s(\tilde{t}_m)s(\tilde{t}_m - 1) < 0$, with $\tilde{t}_{m-1} < \tilde{t}_m$, $\forall m$.

P.3. There are sufficiently many such points $\tilde{t}_m$ between $t_0 + 1$ and t_1, i.e., the known data set contains sufficiently many oscillation cycles.

P.4. $\tilde{t}_m - \tilde{t}_{m-1} \gg 1$, $\forall m$, i.e, there are sufficiently many sample points per oscillation cycle. In general it would be good to have 50-100 points per oscillation cycle.

P.5. $\max\{|s(t)|; 1 \le t \le t_1\} \approx 1$. The available data set is finite, and thus bounded. Thus the amplitude can be rescaled to ≈ 1.

3.3 Implementation for ANN Prediction

In this section, we will provide details concerning the implementation of LTMSP using ANNs[1]. In the next chapter, the test results using the proposed procedure will be reported.

A 2LFF1SNN architecture, represented in Fig.3.1, is used in our study. The first layer TF is chosen to be the hyperboilc tangent sigmoid $f^{(1)}(x) = \sigma(x) = \tanh(x)$ in all experiments. This is the most common choice for TF in ANN applications, due to the smoothness and monotonicity properties of $\tanh(\cdot)$, which facilitate the gradient-based network training. The outputs of the first layer of this type of ANN are given by

$$y^{(1)}_{\mathbf{w},k_1}(t) = \tanh\{v^{(1)}_{\mathbf{w},k_1}(t)\} = \tanh\left\{w^{(1)}_{k_1,0} + \sum_{k_0=1}^{n_0} w^{(1)}_{k_1,k_0} s(t-k_0)\right\} \tag{3.26}$$

for $1 \le k_1 \le n_1$.

The most common choice of the second-layer TF is $f^{(2)}(x) = x$. Most universal approximation results have been proven for such a 2LFF1S2LNN [31, 32, 33, 34, 53, 75, 76, 77, 96, 97, 105]. The network output in that case is given by

$$y^{(2)}_{\mathbf{w}}(t) = w^{(2)}_0 + \sum_{k_1=1}^{n_1} w^{(2)}_{k_1} y^{(1)}_{\mathbf{w},k_1}(t) \tag{3.27}$$

where $y^{(1)}_{\mathbf{w},k_1}(t)$, $1 \le k_1 \le n_1$, are given by (3.26). This is the classical 2LFFNN used in most applications.

As mentioned before, the universal approximation property still holds for 2LFF1SNNs with continuous and monotone sigmoidal transfer function (such as $\tanh(\cdot)$) in the output layer (2LFF1S2SNNs) [27]. Therefore an alternative choice for the second-layer TF could be $f^{(2)}(x) = \tanh(x)$. Since the amplitude of the limit cycle of a given trajectory might be larger than 1, a scaling is introduced so that the ANN is capable of accurately reconstructing this asymptotic state. Hence, $f^{(2)}(x) = \psi(w^{(3)})\tanh(x)$ is proposed as TF in the second layer, where $w^{(3)}$ is an output scaling parameter that will be determined by ANN training, and

$$\psi(x) \overset{\mathrm{def}}{=} \ln\left(1 + \frac{e^x}{1+e^{-x}}\right). \tag{3.28}$$

---

[1] Versions of some of the discussions in this section have been published in [238, 239, 240, 241] or submitted for publication in [241].

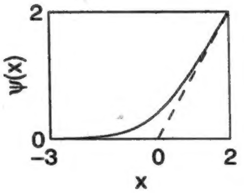

Figure 3.2: The function $\psi(x) = \ln\{1 + \mathrm{e}^{x}/(1 + \mathrm{e}^{-x})\}$ ('—').

The ANN output in that case will be

$$y_{\mathbf{w}}^{(2)}(t) = \psi(w^{(3)}) \tanh \left\{ w_0^{(2)} + \sum_{k_1=1}^{n_1} w_{k_1}^{(2)} y_{\mathbf{w},k_1}^{(1)}(t) \right\}. \tag{3.29}$$

Note that $\psi(x) > \max\{x, 0\}$, $\forall x$, and $\psi(\cdot)$ has $y = 0$, $y = x$ as asymptotes at $-\infty$, $+\infty$ respectively (see Fig.3.2). $\psi(w^{(3)})$ is used instead of $w^{(3)}$ simply to keep the scaling constant positive throughout the ANN training. Clearly, the vector $\mathbf{w}$ will contain the extra parameter $w^{(3)}$ in that case. In addition, $\psi'(x) \approx 1$ for most positive values of x, hence $w^{(3)}$ will vary linearly during network training [240].

The hyperbolic tangent sigmoid is expected to have a noise reduction effect in feature extraction [86], hence by using such a transfer function in the output layer rather than a linear function, we are hoping to achieve a better robustness for applications to noisy data. Our experiments have shown that this does indeed happen. The initial value of $w^{(3)}$ in all experiments (when this type of TF was used) was set to $\psi^{-1}(1)$.

A 2LFF1S2LNN with scaled weights in the second layer is used in some experiments. Namely, all $w_{k_1}^{(2)}$, $1 \le k_1 \le n_1$, are replaced by

$$\tilde{w}_{k_1}^{(2)} \stackrel{\text{def}}{=} \frac{\psi(w^{(3)})\, w_{k_1}^{(2)}}{\sqrt{1 + \sum_{h_1=1}^{n_1} \left[w_{h_1}^{(2)} \right]^2}} \tag{3.30}$$

The ANN output in that case will be

$$y_{\mathbf{w}}^{(2)}(t) = w_0^{(2)} + \sum_{k_1=1}^{n_1} \frac{\psi(w^{(3)})\, w_{k_1}^{(2)}}{\sqrt{1 + \sum_{h_1=1}^{n_1} \left[w_{h_1}^{(2)} \right]^2}}\, y_{\mathbf{w},k_1}^{(1)}(t). \tag{3.31}$$

Note that $\psi(w^{(3)})$ is again a scaling parameter used to adjust the magnitude of the ANN output in order to make it possible for limit cycles of large amplitude to be predicted. Since by (3.26) we have $|y_{\mathbf{w},k_1}^{(1)}(t)| < 1$, it follows, by the Cauchy inequality, that:

$$\left| \frac{y_{\mathbf{w}}^{(2)}(t) - w_0^{(2)}}{\psi(w^{(3)})} \right|^2 < \frac{\left[\sum_{k_1=1}^{n_1} |w_{k_1}^{(2)}| \right]^2}{1 + \sum_{h_1=1}^{n_1} \left[w_{h_1}^{(2)} \right]^2} \le \frac{\left[\sum_{k_1=1}^{n_1} 1^2 \right] \sum_{k_1=1}^{n_1} \left[w_{k_1}^{(2)} \right]^2}{1 + \sum_{h_1=1}^{n_1} \left[w_{h_1}^{(2)} \right]^2} < n_1. \tag{3.32}$$

Hence

$$|y^{(2)}_{\mathbf{w}}(t)| < |w^{(2)}_0| + \psi(w^{(3)})\sqrt{n_1}. \tag{3.33}$$

If $\psi(w^{(3)})$ was mising, the ANN output would be unable to predict limit cycles of amplitude larger than $|w^{(2)}_0| + \sqrt{n_1}$. The vector $\mathbf{w}$ now contains the parameter $w^{(3)}$, and the initial value of $w^{(3)}$ is set to $\psi^{-1}(1)$.

By normalizing the weights in the second layer, we expect to improve the ANN robustness with respect to the noise and to variations in different parameters such as the number of inputs and the number of hidden neurons. Since the second-layer weight updates during network training are interdependent, these weights cannot vary wildly. It is well known that one of the features that provides the power of ANNs is the redundancy in the system parameters. However, in practice, redundancy also leads to overfitting and poor generalization due to the existence of many degrees-of-freedom in the system. One possible way to reconcile these two sides of redundancy could be the weight normalization.

The ANN training performs the minimization of the mean-square OSP error over the training set, given by (3.14) in Section 3.2, using a Fletcher-Reeves conjugate gradient algorithm, as described in Section 2.2. First an initial guess $\mathbf{w}_0$ for the weight vector $\mathbf{w}$ is set. At each training iteration, a sweep of the training set is performed and $\mathbf{g}^{\text{new}} \overset{\text{def}}{=} \nabla E(\mathbf{w}^{\text{old}})$ is computed using the current weight vector $\mathbf{w}^{\text{old}}$. The current search direction is computed: $\mathbf{p}^{\text{new}} = -\mathbf{g}^{\text{new}} + \{||\mathbf{g}^{\text{new}}||_2^2/||\mathbf{g}^{\text{old}}||_2^2\}\mathbf{p}^{\text{old}}$, and finally the weights are updated: $\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + \alpha_{\mathbf{w}}\mathbf{p}^{\text{new}}$, where $\alpha_{\mathbf{w}}$ is the current step size, or the *learning rate (LR)*, which can be constant or adjusted at each iteration. The search direction was set to $\mathbf{p}^{\text{new}} = -\mathbf{g}^{\text{new}}$ at the beginning of training and after every 100 iterations, in order to accelerate the convergence of the conjugate gradient method [86].

The gradient of the performance index at each training iteration is computed as follows. The performance index can be viewed as a sum:

$$E(\mathbf{w}) = \sum_{t=t_0+1}^{t_1} E(\mathbf{w},t), \text{ where } E(\mathbf{w},t) \overset{\text{def}}{=} \frac{[\tilde{e}_{\mathbf{w}}(t)]^2}{t_1 - t_0} \tag{3.34}$$

and therefore

$$\nabla E(\mathbf{w}) = \sum_{t=t_0+1}^{t_1} \nabla E(\mathbf{w},t), \text{ where } \nabla E(\mathbf{w},t) = \frac{-2\tilde{e}_{\mathbf{w}}(t)}{t_1 - t_0}\nabla \tilde{y}_{\mathbf{w}}(t) \tag{3.35}$$

and $\tilde{y}_{\mathbf{w}}(t) = y^{(2)}_{\mathbf{w}}(t)$ is given by

$$y^{(2)}_{\mathbf{w}}(t) = \psi_\xi(w^{(3)})\varphi\left\{ w^{(2)}_0 + \sum_{k_1=1}^{n_1} \frac{\psi_\zeta(w^{(3)})\; w^{(2)}_{k_1}}{\sqrt{1 + \zeta \sum_{h_1=1}^{n_1} \left[w^{(2)}_{h_1}\right]^2}}\, y^{(1)}_{\mathbf{w},k_1}(t) \right\}. \tag{3.36}$$

where $y^{(1)}_{\mathbf{w},k_1}(t)$, $1 \le k_1 \le n_1$, are given by (3.26). The argument of φ (the *net input* to the second layer) will be denoted by $v^{(2)}_{\mathbf{w}}(t)$. We define $\psi_\tau(x) \overset{\text{def}}{=} (1-\tau) + \tau\psi(x)$, $\forall x \in \mathbf{R}$, $\forall \tau \in [0,1]$, so that $\psi_1(x) = \psi(x)$ and $\psi_0(x) = 1$. For $\varphi(x) = x$, $\xi = 0$, $\zeta = 0$ in (3.36), we obtain (3.27); for $\varphi(x) = x$, $\xi = 0$, $\zeta = 1$, we get (3.31); and for $\varphi(x) = \tanh(x)$, $\xi = 1$, $\zeta = 0$, we obtain (3.29).

At each training iteration, the scaled weights are first computed:

$$w^{(2,\mathrm{sc})}_{k_1} \overset{\text{def}}{=} \frac{w^{(2)}_{k_1}}{r^{(2)}} \overset{\text{def}}{=} \frac{w^{(2)}_{k_1}}{\sqrt{1 + \zeta \sum\limits_{h_1=1}^{n_1} \left[w^{(2)}_{h_1}\right]^2}}, \quad 1 \le k_1 \le n_1. \tag{3.37}$$

Then, for every t, $t_0 + 1 \le t \le t_1$, the values of $v^{(1)}_{\mathbf{w},k_1}(t)$, $y^{(1)}_{\mathbf{w},k_1}(t)$, $1 \le k_1 \le n_1$, $v^{(2)}_{\mathbf{w}}(t)$, $y^{(2)}_{\mathbf{w}}(t)$, and $\tilde{e}_{\mathbf{w}}(t) = s(t) - y^{(2)}_{\mathbf{w}}(t)$ are calculated according to (3.26) and (3.36). The following quantities are then computed using the backpropagation algorithm:

$$\frac{\partial E(\mathbf{w},t)}{\partial y^{(2)}_{\mathbf{w}}(t)} = \frac{-2\tilde{e}_{\mathbf{w}}(t)}{t_1 - t_0}, \quad \frac{\partial E(\mathbf{w},t)}{\partial v^{(2)}_{\mathbf{w}}(t)} = \frac{\partial E(\mathbf{w},t)}{\partial y^{(2)}_{\mathbf{w}}(t)} \psi_\xi(w^{(3)}) \dot{\varphi}\{v^{(2)}_{\mathbf{w}}(t)\}, \tag{3.38}$$

$$\frac{\partial E(\mathbf{w},t)}{\partial y^{(1)}_{\mathbf{w},k_1}(t)} = \frac{\partial E(\mathbf{w},t)}{\partial v^{(2)}_{\mathbf{w}}(t)} \psi_\zeta(w^{(3)}) w^{(2,\mathrm{sc})}_{k_1}, \quad \frac{\partial E(\mathbf{w},t)}{\partial v^{(1)}_{\mathbf{w},k_1}(t)} = \frac{\partial E(\mathbf{w},t)}{\partial y^{(1)}_{\mathbf{w},k_1}(t)} \dot{\sigma}(v^{(1)}_{\mathbf{w},k_1}(t)), \tag{3.39}$$

where $\sigma(x) = \tanh(x)$. Then the gradient components of $E(\mathbf{w},t)$ are computed as follows.

$$\frac{\partial E(\mathbf{w},t)}{\partial w^{(1)}_{k_1,0}} = \frac{\partial E(\mathbf{w},t)}{\partial v^{(1)}_{\mathbf{w},k_1}(t)}, \quad \frac{\partial E(\mathbf{w},t)}{\partial w^{(1)}_{k_1,k_0}} = \frac{\partial E(\mathbf{w},t)}{\partial v^{(1)}_{\mathbf{w},k_1}(t)} s(t - k_0), \ 1 \le k_0 \le n_0, \tag{3.40}$$

Note that, for $1 \le k_1 \le n_1$:

$$\frac{\partial v^{(2)}_{\mathbf{w}}(t)}{\partial w^{(2)}_{k_1}} = \frac{\psi_\zeta(w^{(3)})}{\left[r^{(2)}\right]^2} \left\{ y^{(1)}_{\mathbf{w},k_1}(t) r^{(2)} - \left[\sum_{h_1=1}^{n_1} w^{(2)}_{h_1} y^{(1)}_{\mathbf{w},h_1}(t)\right] \frac{w^{(2)}_{k_1}}{r^{(2)}} \right\} \tag{3.41}$$

or, equivalently:

$$\frac{\partial v^{(2)}_{\mathbf{w}}(t)}{\partial w^{(2)}_{k_1}} = \frac{1}{\left[r^{(2)}\right]^2} \left\{ \psi_\zeta(w^{(3)}) y^{(1)}_{\mathbf{w},k_1}(t) r^{(2)} - \left[v^{(2)}_{\mathbf{w}}(t) - w^{(2)}_0\right] w^{(2)}_{k_1} \right\}. \tag{3.42}$$

Hence

$$\frac{\partial E(\mathbf{w},t)}{\partial w^{(2)}_0} = \frac{\partial E(\mathbf{w},t)}{\partial v^{(2)}_{\mathbf{w}}(t)}, \quad \frac{\partial E(\mathbf{w},t)}{\partial w^{(2)}_{k_1}} = \frac{\partial E(\mathbf{w},t)}{\partial v^{(2)}_{\mathbf{w}}(t)} \frac{\partial v^{(2)}_{\mathbf{w}}(t)}{\partial w^{(2)}_{k_1}}, \ 1 \le k_1 \le n_1. \tag{3.43}$$

Finally:

$$\frac{\partial E(\mathbf{w},t)}{\partial w^{(3)}} = \frac{\partial E(\mathbf{w},t)}{\partial y^{(2)}_{\mathbf{w}}(t)} \frac{\dot{\psi}_\xi(w^{(3)})}{\psi_\xi(w^{(3)})} y^{(2)}_{\mathbf{w}}(t) + \frac{\partial E(\mathbf{w},t)}{\partial v^{(2)}_{\mathbf{w}}(t)} \frac{\dot{\psi}_\zeta(w^{(3)})}{\psi_\zeta(w^{(3)})} \left[v^{(2)}_{\mathbf{w}}(t) - w^{(2)}_0\right]. \tag{3.44}$$

Designing a good stopping criterion for the network training process is of utmost importance in ANN applications. The purpose of training is for the network to learn the training patterns, in other words to make the performance index $E(\mathbf{w})$ small. However, the learning of the training examples is desirable to the extent to which it helps the ANN to accurately generalize. If $E(\mathbf{w})$ becomes smaller than a certain threshold, the ANN will start to learn features that are specific to the training set and are not relevant anymore for the general pattern from which the training examples were drawn. This threshold is heavily problem-dependent and there is no clear way to estimate it. In our study we use a stopping criterion that is specific to our MSP problem. At each iteration, after the weight update has been performed, the current MSP vector

$$\hat{\mathbf{y}}(\mathbf{w}^{\text{new}}) \overset{\text{def}}{=} [\hat{y}_{\mathbf{w}^{\text{new}}}(t_1+1), \hat{y}_{\mathbf{w}^{\text{new}}}(t_1+2), \ldots, \hat{y}_{\mathbf{w}^{\text{new}}}(t_2)]^T \tag{3.45}$$

is computed. The ANN training is stopped when the distance between two consecutive MSP-generated signals

$$\rho^{\text{new}} \overset{\text{def}}{=} \frac{||\hat{\mathbf{y}}(\mathbf{w}^{\text{new}}) - \hat{\mathbf{y}}(\mathbf{w}^{\text{old}})||_2}{\sqrt{t_2 - t_1}} \tag{3.46}$$

becomes small and almost constant during many training iterations. If this does not happen, then we say that the ANN training does not converge [238, 239]. In our experiments, the value of $E(\mathbf{w}^{\text{new}})$ is usually around 0.01 and the value of ρ^{new} is of the order of 10^{-4} by the time the ANN training converges.

As mentioned above, the LR can be constant or adaptive. For a given test case, if a constant learning rate (CLR) is used, its value is chosen as follows. Starting with an initial value of 1.5, decrement the CLR with 0.1 at a time and start the ANN training. Then the CLR is chosen to be 1/10 of the maximum value for which the ANN training remains stable during the first 1000 iterations.

In the case of an adaptive LR (ALR) α_{w}, the training is performed as follows. At each iteration, after the weight update, α_{w} is multiplied by $\zeta_{\text{w}} = 1.1$ if $E(\mathbf{w}^{\text{new}}) < E(\mathbf{w}^{\text{old}})$, and divided by ζ_{w} otherwise. In order for α_{w} not to decay to zero, a constant lower bound $\hat{\alpha}_{\text{w}}$ for α_{w} is set to 10^{-6}. At the beginning of training, α_{w} is set to $\hat{\alpha}_{\text{w}}$. To avoid a heavy oscillatory behavior of the learning rate, an adaptive upper bound α^*_{w} for α_{w} is initialized. Then, whenever α_{w} decreases, α^*_{w} is divided by $\zeta^*_{\text{w}} = 1.001$, otherwise it remains unchanged. As a result, both α_{w} and α^*_{w} eventually stabilize at the same value α^0_{w} [238, 239]. We select the initial value of α^*_{w} as follows. Starting with 0.1, increment the initial α^*_{w} with 0.1 at a time and start the ANN training. Choose the minimum upper bound for which the ALR has at least one oscillation within the first 1000 training iterations. Using an ALR is expected to increase the sensitivity of the ANN to noise in the training signal, due to the fact that an ALR search algorithm is more versatile than one using a CLR, and therefore makes it possible for the ANN to learn the noise immediately, thus corrupting the underlying pattern.

The standard approach to weight initialization for training is to set the weights to small random values. It is easy to see that $\mathbf{w} = \mathbf{0}$ is a steady state for the ANN training process. If the training is started with zero initial weights, the gradient of the performance index is zero, and hence the weight updates are zero. Therefore $\mathbf{w}^{\text{new}} = \mathbf{0}$ at every training iteration. In some of our experiments we will initialize all $w^{(1)}_{k_1,k_0}, k_1 \neq 0, k_0 \neq 0$ with random numbers uniformly distributed in $[-0.01, +0.01]$, while all the other weights will be initially set to zero. This is, in fact, sufficient to ensure a nonzero gradient at the beginning of training. In other experiments, $w^{(1)}_{k_1,k_0}, k_1 \neq 0, k_0 \neq 0$, are initially set to

$$w^{(1)}_{k_1,k_0} := \frac{s(t_1 - k_1 - k_0 + 1)}{\sum_{h_0=1}^{n_0} [s(t_1 - k_1 - h_0 + 1)]^2}, \tag{3.47}$$

while all the other weights are initialized by zero. The reason for doing that will be explained in the following. It was noticed in practice that, if the network training is performed with zero initial weights in the first layer (and nonzero initial weights in the second layer), then each vector $\mathbf{w}^{(1)}_{k_1,\cdot}$ has an oscillatory profile at the end of training. Moreover, the main oscillation frequency is close to the one corresponding to the training signal. This phenomenon could be explained by the fact that

$$\frac{\partial E(\mathbf{w},t)}{\partial w^{(1)}_{k_1,k_0}} = \frac{\partial E(\mathbf{w},t)}{\partial w^{(1)}_{k_1,0}}\, s(t - k_0) \tag{3.48}$$

for $t_0 + 1 \leq t \leq t_1$, $1 \leq k_1 \leq n_1$, $1 \leq k_0 \leq n_0$. In the steepest descent optimization algorithm, for instance, the weight update at each iteration is proportional to $\nabla E(\mathbf{w})$, and for each t, the gradient components corresponding to the weights of a first-layer neuron k_1 are proportional to the inputs $s(t - k_0)$. Therefore, it is not surprising that $\mathbf{w}^{(1)}_{k_1,\cdot}$ has an oscillatory profile. In the conjugate gradient algorithm, the current weight update depends on a linear combination of the gradient of the performance index at several past iterations, but the above proportionality still holds.

This observation leads to the idea that one might be able to speed up the ANN training by initializing the vectors $\mathbf{w}^{(1)}_{k_1,\cdot}$ with segments of the training set, thus incorporating an oscillation in the weights from the very beginning of the training. In our experiments, however, we did not notice any significant improvement in the training speed when such a weight initialization was used, but a clear improvement in prediction accuracy and robustness was observed. The division of each segment by its squared Euclidian norm ('normalization') was done in order to prevent the net input $v^{(1)}_{\mathbf{w},k_1}(t)$ of each hidden neuron from being large in absolute value, which would then lead to the phenomenon of saturation. Since the derivative of $\tanh(\cdot)$ is close to zero even for $|x|$ close to 2, the derivatives of the performance index $E(\mathbf{w})$ with respect to the weights of the corresponding neuron would be close to zero. Consequently, the weights would never be updated by the gradient-based ANN training algorithm.

Once the network architecture and the training algorithm are determined, the remaining important consideration is to choose the correct number of network inputs n_0 and the number of neurons in the hidden layer n_1 for a given task. In the present study, we propose various methods that provide estimates for n_0 and n_1. These methods are extensions of those we proposed in [240].

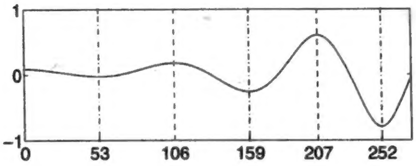

Figure 3.3: ANN Training Set with Lmax and Lmin Points (Test Signal S1).

Since ANNs are robust with respect to variations in the number of inputs and in the number of neurons, rough estimates for these quantities should be sufficient.

The main idea when estimating n_0 is that we want the ANN inputs to span over an entire oscillation cycle. A first estimate of n_0 could therefore be obtained by dividing the number of training points $t_1 - t_0$ by the number of oscillation cycles observed in the training set, thus obtaining an empirical estimate of the average number of sample points per cycle [240]. Alternatively, we could choose n_0 to be the average length of an oscillation cycle, namely the average distance between consecutive local minimum (Lmin) — or local maximum (Lmax) — points throughout the training set. For example, in Fig.3.3, $t_1^{\text{Lmin}} = 53$, $t_2^{\text{Lmin}} = 159$, $t_3^{\text{Lmin}} = 252$ are the Lmin points, while $t_1^{\text{Lmax}} = 106$, $t_2^{\text{Lmax}} = 207$ are the Lmax points. The distances between consecutive Lmin points are $d_1^{\text{Lmin}} = t_2^{\text{Lmin}} - t_1^{\text{Lmin}} = 106$, $d_2^{\text{Lmin}} = t_3^{\text{Lmin}} - t_2^{\text{Lmin}} = 93$, while the distance between consecutive Lmax points is $d_1^{\text{Lmax}} = t_2^{\text{Lmax}} - t_1^{\text{Lmax}} = 101$. The average distance between the consecutive Lmin points is thus $a^{\text{Lmin}} = 100$ (rounded off to the closest integer), while the average distance between the consecutive Lmax points is $a^{\text{Lmax}} = 101$. The average of these two quantities, $a^{\text{avg}} = 101$, could be an empirical estimate, denoted by n_0^{emp}, of n_0. Note that the following vectors are defined: $\mathbf{t}^{\text{Lmin}} \overset{\text{def}}{=} \left[t_1^{\text{Lmin}}, t_2^{\text{Lmin}}, t_3^{\text{Lmin}}\right]^T = [53, 159, 252]^T$, $\mathbf{d}^{\text{Lmin}} \overset{\text{def}}{=} \left[d_1^{\text{Lmin}}, d_2^{\text{Lmin}}\right]^T = [106, 93]^T$, $\mathbf{t}^{\text{Lmax}} \overset{\text{def}}{=} \left[t_1^{\text{Lmax}}, t_2^{\text{Lmax}}\right]^T = [106, 207]^T$, $\mathbf{d}^{\text{Lmax}} \overset{\text{def}}{=} \left[d_1^{\text{Lmax}}\right]^T = [101]^T$.

An alternative way of determining an appropriate value for n_0 is to compute, for $0 \leq h \leq t_1 - t_0 - 1$, an estimate of the autocorrelation function (ACF) [68, 215]:

$$\hat{\rho}(h) = \frac{\sum_{t=t_0+1}^{t_1-h} [s(t+h) - \bar{s}][s(t) - \bar{s}]}{\sum_{t=t_0+1}^{t_1} [s(t) - \bar{s}]^2}, \text{ where } \bar{s} = \frac{1}{t_1 - t_0} \sum_{t=t_0+1}^{t_1} s(t). \tag{3.49}$$

According to Fan and Yao (citing Box and Jenkins) [68], in order for the estimates to be accurate it is necessary to have $t_1 - t_0 \geq 50$ — which is always satisfied in our case studies. Moreover, since the estimates become inaccurate for larger h because of the lack of enough observations, it has been proposed that only the values of $\hat{\rho}(h)$ for $0 \leq h \leq (t_1 - t_0)/4$ be used [68].

Note that the ACF is a measure of the **linear** dependence of $s(t)$ on $s(t-h)$ for different lags h [68, 215]. In our experiments, the function $\hat{\rho}(\cdot)$ has an oscillatory profile, as in Fig.3.4, and $|\hat{\rho}(h)|$ is typically a nondecreasing function of h. The value of h for which the first peak of $\hat{\rho}(h)$ is achieved provides an estimate n_0^{acf} of the lag h for which the positive correlation between $s(t)$

and $s(t-h)$, $t>0$, is maximum. For instance, in Fig.3.4, the first peak is achieved for $h=43$, hence $n_0^{\mathrm{acf}}=43$ is an estimate of the necessary number of ANN inputs. The vertical bar marks the point $h=(t_1-t_0)/4$.

An estimate of the number of network inputs can also be obtained as follows. Compute the Fast Fourier Transform (FFT) of the training set assuming the sampling step to be 1. Detect the lowest frequency ν_1 that has an energy close to the maximum energy of all Fourier modes in the decomposition. Compute $n_0^{\mathrm{fft}}=1/\nu_1$ (rounding off to an integer value). This is an estimate of the number of sample points contained in the oscillation period T_1 of the lowest frequency mode with high energy. For instance, in Fig.3.5, $\nu_1=0.0105$ does not have the highest energy, but it is the lowest frequency that has an energy close to the highest one in the decomposition. Therefore we take $n_0^{\mathrm{fft}}=1/\nu_1\approx 95$ as an estimate of the number of network inputs necessary for working with this signal.

In the next chapter, we show that the estimates n_0^{emp}, n_0^{acf}, and n_0^{fft} are close to each other for each test case (see Table 4.4). Hence, the "final" estimate n_0^{fin} of n_0 could be chosen to be either the maximum n_0^{max} of the three types of estimates, or their average n_0^{avg}, possibly rounded up to the closest multiple of 10, etc. Our choice of n_0^{fin} for each test case will be reported in Table 4.4.

It was noticed in practice that quantitative pointwise prediction error measures do not provide a good description of the MSP accuracy. Even when a small phase shift between the correct signal and the predicted signal is present, the values of the pointwise error could be of the same order as the amplitude of the given signal. Therefore we need to establish qualitative criteria for assessing the prediction accuracy. An obvious way is to determine the degree of superposition between the time histories (THs) of the correct signal and the predicted signal by visual inspection. Additional insight can be obtained by plotting $x(t)$ vs. $x(t-d)$ — where t spans over a complete oscillation cycle — for both the correct signal and the predicted signal, for different lags $d=1$, 2, etc, and assessing the degree of superposition between the two graphs for each time lag. Since we are only interested in accurately reconstructing the asymptotic state of the signal (the limit cycle, in our test cases), we could plot n_0 graphs for each test case, namely for $1\le d\le n_0$, $t_2-n_0\le t\le t_2$. Due to space limitations, throughout this thesis we will only plot the phase portraits (PPs) for selected lags d. For instance, in Fig.3.6, the superposition of the THs of the test signal and the ANN-generated MSP is illustrated, as well as the superposition between the PPs corresponding

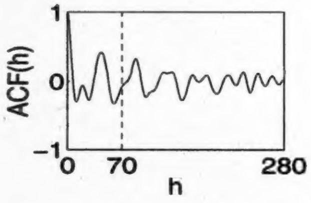

Figure 3.4: ACF for an ANN Training Set (Test Signal S3).

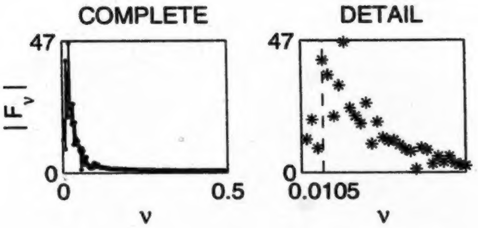

Figure 3.5: FFT Plot of a Training Set (Test Signal S5).

to $d = 3$.

Having chosen the number of inputs n_0, the necessary number of ANN hidden neurons n_1 needs to be estimated. In the present study we propose an extension of the method presented earlier in [240]. In [240] we proposed performing the ANN training for different numbers of hidden neurons $n_1 = 1, 2, 3, \ldots$. This process could be parallelized in order to reduce the computation time. Let $\mathbf{w}^{[n_1]}$ be the weights obtained when training an ANN with n_1 hidden neurons using the given signal, and

$$\rho^{[n_1]} \overset{\text{def}}{=} \frac{\left\|\hat{\mathbf{y}}\left(\mathbf{w}^{[n_1]}\right) - \hat{\mathbf{y}}\left(\mathbf{w}^{[n_1-1]}\right)\right\|_2}{\sqrt{t_2 - t_1}} \tag{3.50}$$

for each $n_1 > 1$, where $\hat{\mathbf{y}}(\mathbf{w})$ is defined in (3.45). In general, the distance $\rho^{[n_1]}$ first decreases as n_1 increases, then it becomes smaller and approximately constant with respect to n_1 (see Fig.3.7(a)). This indicates that the ANN prediction is stable. In order to visualize the overall trend, $\rho^{[\cdot]}$ can be averaged over q consecutive values of n_1, obtaining a distance $\bar{\rho}_q^{[\cdot]}$. One can start with $q = 1$, 2, etc., until the graph of $\bar{\rho}_q^{[\cdot]}$ becomes relatively smooth. The value $\bar{n}_1$ is selected which corresponds to the point where the decreasing trend of $\bar{\rho}_q^{[\cdot]}$ stops, and $\bar{n}_1 + q$ is chosen as the number of hidden neurons. For instance, in Fig.3.7(b), $\bar{\rho}_5^{[\cdot]}$ is considered (for $q = 5$) and the decreasing trend of $\bar{\rho}_5^{[\cdot]}$ stops at $\bar{n}_1 = 29$. Therefore the number of hidden neurons is chosen to be $n_1 = 29 + 5 = 34$.

In fact, we are mainly interested in the asymptotic state of the predicted signal, regardless of the shifts between MSPs corresponding to different numbers of hidden neurons. In Fig.3.8 it is apparent that the PPs of the MSP stabilize much sooner than the THs as the number of hidden neurons is increased. Therefore an alternative choice for n_1 could be the value $\bar{n}_1$ for which the

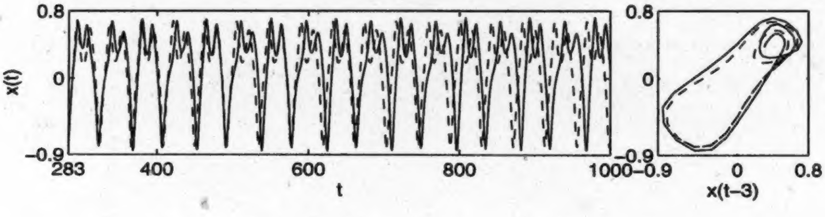

Figure 3.6: Test Signal ('—'), and MSP ('- -').

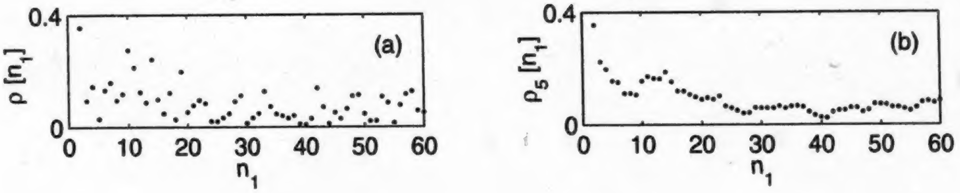

Figure 3.7: Sample $\rho^{[\cdot]}$ (a) and $\bar{\rho}_5^{[\cdot]}$ (b).

PPs of $\hat{\mathbf{y}}\left(\mathbf{w}^{[\bar{n}_1-D]}\right)$, $\hat{\mathbf{y}}\left(\mathbf{w}^{[\bar{n}_1-D+1]}\right)$, ..., $\hat{\mathbf{y}}\left(\mathbf{w}^{[\bar{n}_1-1]}\right)$, $\hat{\mathbf{y}}\left(\mathbf{w}^{[\bar{n}_1]}\right)$ are overlapping for each lag d, $1 \leq d \leq n_0$ (for some integer $D > 0$).

3.4 Dynamical Systems Perspective

In this section, we present a theoretical justification of our proposed approach to LTMSP of nonlinear trajectories using ANNs. We are providing a reason why we expect our method to work. We will show that the MSP dynamical response is close to the response of the original dynamical system when certain conditions are satisfied. More specifically, we compute an upper bound on the distance between the ω-limit set of the given trajectory $\{s(t); t \geq 1\}$ and the ω-limit set of the trajectory generated by MSP using an ANN trained based on a segment of the transient state of $\{s(t); t \geq 1\}$.

Notation 3.4.1. Denote by $\{\mathbf{e}_h; 1 \leq h \leq n_0\}$ the canonical basis in $\mathbf{R}^{n_0}$. For every h, $1 \leq h \leq n_0$: $\mathbf{e}_h = [0, \ldots, 0, 1, 0, \ldots, 0]^T \in \mathbf{R}^{n_0}$, where the h-th component is equal to 1.

Notation 3.4.2. Denote by $\partial_h f$ the partial derivative with respect to the h-th variable of a differentiable mapping $f : \mathbf{R}^{n_0} \to \mathbf{R}$.

Notation 3.4.3. Denote by $C^0(\mathbf{R}^n; \mathbf{R}^m)$ the set of all continuous mappings $f : \mathbf{R}^n \to \mathbf{R}^m$, and by $C_0^0(\mathbf{R}^n; \mathbf{R}^m)$ the subset of all bounded mappings in $C^0(\mathbf{R}^n; \mathbf{R}^m)$.

Notation 3.4.4. Denote by $C^k(\mathbf{R}^n; \mathbf{R}^m)$ the set of all mappings $f : \mathbf{R}^n \to \mathbf{R}^m$ that have continuously differentiable partial derivatives of order up to k. Denote by $C_0^k(\mathbf{R}^n; \mathbf{R}^m)$ the

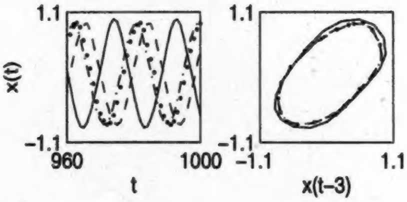

Figure 3.8: MSP for $n_1 = 3$ ('—'), 4('- -'), 5('-.'), 6('...').

subset of bounded functions in $C^k(\mathbf{R}^n;\mathbf{R}^m)$ for which all partial derivatives of order up to k are bounded.

Notation 3.4.5. Denote by $C^\infty(\mathbf{R}^n;\mathbf{R}^m)$ the set of all infinitely differentiable mappings $f : \mathbf{R}^n \to \mathbf{R}^m$. Denote by $C_0^\infty(\mathbf{R}^n;\mathbf{R}^m)$ the subset of bounded functions in $C^\infty(\mathbf{R}^n;\mathbf{R}^m)$ for which all partial derivatives are bounded.

Notation 3.4.6. For $m \in \mathbf{N}_0$, denote by $f^{[m]}$ the m-fold composition $f \circ f \circ \ldots \circ f$ of a function $f : \mathbf{R}^n \to \mathbf{R}^n$. By definition, $f^{[0]}$ is the identity function.

Definition 3.4.1. Define the (open) ball of center $\tilde{\mathbf{x}}$ and radius r with respect to the ∞-norm on $\mathbf{R}^{n_0}$ by $B_\infty(\tilde{\mathbf{x}}, r) \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbf{R}^{n_0}; ||\mathbf{x} - \tilde{\mathbf{x}}||_\infty < r\}$, $\forall \tilde{\mathbf{x}} \in \mathbf{R}^{n_0}$, $\forall r > 0$.

Definition 3.4.2. Define the closed ball of center $\tilde{\mathbf{x}}$ and radius r with respect to the ∞-norm on $\mathbf{R}^{n_0}$ by $\bar{B}_\infty(\tilde{\mathbf{x}}, r) \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbf{R}^{n_0}; ||\mathbf{x} - \tilde{\mathbf{x}}||_\infty \leq r\}$, $\forall \tilde{\mathbf{x}} \in \mathbf{R}^{n_0}$, $\forall r \geq 0$.

Definition 3.4.3. Define the right shift operator $\mathbf{shr} : \mathbf{R}^{n_0} \to \mathbf{R}^{n_0}$ as $\mathbf{shr}(\mathbf{x}) \stackrel{\text{def}}{=} [0, x_1, \ldots, x_{n_0-1}]^T$, $\forall \mathbf{x} \in \mathbf{R}^{n_0}$.

Lemma 3.4.1. $\mathbf{shr}(\cdot)$ is a linear mapping and $||\mathbf{shr}(\mathbf{x})||_\infty \leq ||\mathbf{x}||_\infty$, $\forall \mathbf{x} \in \mathbf{R}^{n_0}$.

Proof. The linearity of $\mathbf{shr}(\cdot)$ is obvious. Moreover, $\forall \mathbf{x} \in \mathbf{R}^{n_0}$, we have:

$$||\mathbf{shr}(\mathbf{x})||_\infty = \max_{1 \leq h \leq n_0 - 1} |x_h| \leq \max_{1 \leq h \leq n_0} |x_h| = ||\mathbf{x}||_\infty. \tag{3.51}$$

□

Definition 3.4.4. Given the mapping $\Phi_\mathbf{w} : \mathbf{R}^{n_0} \to \mathbf{R}$, define the mapping $\mathbf{\Phi_w} : \mathbf{R}^{n_0} \to \mathbf{R}^{n_0}$ by $\mathbf{\Phi_w}(\mathbf{x}) \stackrel{\text{def}}{=} \Phi_\mathbf{w}(\mathbf{x})\mathbf{e}_1 + \mathbf{shr}(\mathbf{x}) = [\Phi_\mathbf{w}(\mathbf{x}), x_1, \ldots, x_{n_0-1}]^T$, $\forall \mathbf{x} \in \mathbf{R}^{n_0}$.

Definition 3.4.5. Given the mapping $\bar{\Phi} : \mathbf{R}^{n_0} \to \mathbf{R}$ in (3.8), define the mapping $\bar{\mathbf{\Phi}} : \mathbf{R}^{n_0} \to \mathbf{R}^{n_0}$ by $\bar{\mathbf{\Phi}}(\mathbf{x}) \stackrel{\text{def}}{=} \bar{\Phi}(\mathbf{x})\mathbf{e}_1 + \mathbf{shr}(\mathbf{x}) = [\bar{\Phi}(\mathbf{x}), x_1, \ldots, x_{n_0-1}]^T$, $\forall \mathbf{x} \in \mathbf{R}^{n_0}$.

Definition 3.4.6. A mapping $\Phi : \mathbf{R}^{n_0} \to \mathbf{R}$ is said to be a Lipschitz mapping if $\exists \lambda(\Phi) > 0$ such that $|\Phi(\tilde{\mathbf{x}}) - \Phi(\hat{\mathbf{x}})| \leq \lambda(\Phi)\ ||\tilde{\mathbf{x}} - \hat{\mathbf{x}}||_\infty$, $\forall \tilde{\mathbf{x}}, \hat{\mathbf{x}} \in \mathbf{R}^{n_0}$.

Definition 3.4.7. A mapping $\mathbf{\Phi} : \mathbf{R}^{n_0} \to \mathbf{R}^{n_0}$ is said to be a Lipschitz mapping if $\exists \lambda(\mathbf{\Phi}) > 0$ such that $||\mathbf{\Phi}(\tilde{\mathbf{x}}) - \mathbf{\Phi}(\hat{\mathbf{x}})||_\infty \leq \lambda(\mathbf{\Phi})\ ||\tilde{\mathbf{x}} - \hat{\mathbf{x}}||_\infty$, $\forall \tilde{\mathbf{x}}, \hat{\mathbf{x}} \in \mathbf{R}^{n_0}$.

Definition 3.4.8. Given a mapping $\mathbf{\Phi} : X \to X$, a subset $Y \subset X$ is said to be *forward* $\mathbf{\Phi}$*-invariant* if $\mathbf{\Phi}(Y) \subset Y$.

Assumption 3.4.1. Throughout this section, we assume that the mapping $\Phi_\mathbf{w} : \mathbf{R}^{n_0} \to \mathbf{R}$ is given by the output of a 2LFFNN:

$$\Phi_\mathbf{w}(\mathbf{x}) \stackrel{\text{def}}{=} f^{(2)} \left\{ w_0^{(2)} + \sum_{k=1}^{n_1} w_k^{(2)} f^{(1)} \left(w_{k,0}^{(1)} + \sum_{h=1}^{n_0} w_{k,h}^{(1)} x_h \right) \right\}, \quad \forall \mathbf{x} \in \mathbf{R}^{n_0}. \tag{3.52}$$

with $f^{(1)}(x) = \tanh(x)$, and $f^{(2)}(x) = x$ or $f^{(2)}(x) = c\tanh(x)$ $(c > 0)$.

Proposition 3.4.1. $f^{(1)} \in C_0^\infty(\mathbf{R};\mathbf{R})$, with $|||f^{(1)}|||_C = 1$, $|||\dot{f}^{(1)}|||_C = 1$, and $f^{(2)} \in C^\infty(\mathbf{R};\mathbf{R})$, with $\dot{f}^{(2)} \in C_0^\infty(\mathbf{R};\mathbf{R})$, $|||\dot{f}^{(2)}|||_C = 1$ (if $f^{(2)}(x) = x$) or $|||\dot{f}^{(2)}|||_C = c$ (if $f^{(2)}(x) = c\tanh(x)$, $c > 0$) respectively.

Proof. Follows immediately from Assumption 3.4.1, since $\partial \tanh(x)/\partial x = 1 - \tanh^2(x)$. ☐

Lemma 3.4.2. If $f \in C_0^1(\mathbf{R};\mathbf{R})$, then $|f(\tilde{x}) - f(\hat{x})| \leq |||\dot{f}|||_C \, ||\tilde{x} - \hat{x}||_\infty$, $\forall \tilde{x}, \hat{x} \in \mathbf{R}$.

Proof. Follows immediately from the mean value formula in $\mathbf{R}$. ☐

Proposition 3.4.2. $\Phi_{\mathbf{w}}$ is a Lipschitz mapping, with $\lambda(\Phi_{\mathbf{w}}) \overset{\mathrm{def}}{=} |||\dot{f}^{(2)}|||_C \sum_{k=1}^{n_1} |w_k^{(2)}| \sum_{h=1}^{n_0} |w_{k,h}^{(1)}|$.

Proof. Follows immediately from Assumption 3.4.1 and Lemma 3.4.2, taking into account that $|||\dot{f}^{(1)}|||_C = 1$, as shown by Proposition 3.4.1. ☐

Proposition 3.4.3. $|||\partial_h \Phi_{\mathbf{w}}|||_C \leq |||\dot{f}^{(2)}|||_C \sum_{k=1}^{n_1} |w_k^{(2)}| \, |w_{k,h}^{(1)}|$, $1 \leq h \leq n_0$.

Proof. Follows immediately from Assumption 3.4.1 and Lemma 3.4.2, taking into account that $|||\dot{f}^{(1)}|||_C = 1$, as shown by Proposition 3.4.1. ☐

Proposition 3.4.4. $|\hat{e}_{\mathbf{w}}(t) - \tilde{e}_{\mathbf{w}}(t)| \leq \lambda(\Phi_{\mathbf{w}}) \, ||\hat{\mathbf{e}}_{\mathbf{w}}(t-1)||_\infty$, $\forall t \geq t_1 + 1$.

Proof. By Proposition 3.4.2: $|\hat{e}_{\mathbf{w}}(t) - \tilde{e}_{\mathbf{w}}(t)| = |\tilde{y}_{\mathbf{w}}(t) - \hat{y}_{\mathbf{w}}(t)| = |\Phi_{\mathbf{w}}(\mathbf{s}(t-1)) - \Phi_{\mathbf{w}}(\hat{\mathbf{y}}_{\mathbf{w}}(t-1))| \leq \lambda(\Phi_{\mathbf{w}})||\mathbf{s}(t-1) - \hat{\mathbf{y}}_{\mathbf{w}}(t-1)||_\infty = \lambda(\Phi_{\mathbf{w}})||\hat{\mathbf{e}}_{\mathbf{w}}(t-1)||_\infty$. ☐

Proposition 3.4.5. $|||\Phi_{\mathbf{w}}|||_C \leq \mu(\Phi_{\mathbf{w}}) \overset{\mathrm{def}}{=} |||\dot{f}^{(2)}|||_C \left\{ |w_0^{(2)}| + \sum_{k=1}^{n_1} |w_k^{(2)}| \right\}$.

Proof. Follows immediately from (3.52) and Lemma 3.4.2, taking into account that $f^{(2)}(0) = 0$ for our choices of $f^{(2)}$ and that $|||f^{(1)}|||_C = 1$, as shown by Proposition 3.4.1. ☐

Proposition 3.4.6. $\boldsymbol{\Phi}_{\mathbf{w}}$ is a Lipschitz mapping with $\lambda(\boldsymbol{\Phi}_{\mathbf{w}}) \overset{\mathrm{def}}{=} \max\{1, \lambda(\Phi_{\mathbf{w}})\}$.

Proof. By Definition 3.4.4, Lemma 3.4.1, and Proposition 3.4.2, $\forall \tilde{\mathbf{x}}, \hat{\mathbf{x}} \in \mathbf{R}^{n_0}$ we have:

$$
\begin{aligned}
||\boldsymbol{\Phi}_{\mathbf{w}}(\tilde{\mathbf{x}}) - \boldsymbol{\Phi}_{\mathbf{w}}(\hat{\mathbf{x}})||_\infty &= \max\{|\Phi_{\mathbf{w}}(\tilde{\mathbf{x}}) - \Phi_{\mathbf{w}}(\hat{\mathbf{x}})|, ||\mathbf{shr}(\tilde{\mathbf{x}} - \hat{\mathbf{x}})||_\infty\} \leq \\
&\leq \max\{\lambda(\Phi_{\mathbf{w}})||\tilde{\mathbf{x}} - \hat{\mathbf{x}}||_\infty, ||\tilde{\mathbf{x}} - \hat{\mathbf{x}}||_\infty\} = \max\{1, \lambda(\Phi_{\mathbf{w}})\}||\tilde{\mathbf{x}} - \hat{\mathbf{x}}||_\infty.
\end{aligned} \tag{3.53}
$$

☐

Proposition 3.4.7. $||\boldsymbol{\Phi}_{\mathbf{w}}(\mathbf{x})||_\infty \leq \max\{|\Phi_{\mathbf{w}}(\mathbf{x})|, ||\mathbf{x}||_\infty\} \leq \max\{\mu(\Phi_{\mathbf{w}}), ||\mathbf{x}||_\infty\}$, $\forall \mathbf{x} \in \mathbf{R}^{n_0}$.

Proof. Follows immediately from Definition 3.4.4, Lemma 3.4.1, and Proposition 3.4.5. ☐

Proposition 3.4.8. If $||\mathbf{x}||_\infty \le R$ (for some $R > 0$) then

$$|\Phi_{\mathbf{w}}(\mathbf{x})| \le |||f^{(2)}|||_C \left\{ |w_0^{(2)}| + \mu(R) \sum_{k=1}^{n_1} |w_k^{(2)}| \right\} < \mu(\Phi_{\mathbf{w}}) \tag{3.54}$$

where $\mu(R) = \tanh(\nu(R)) < 1$ with

$$\nu(R) \overset{\text{def}}{=} \max \left\{ |w_{k,0}^{(1)}| + R \sum_{h=1}^{n_0} |w_{k,h}^{(1)}| \; ; \; 1 \le k \le n_1 \right\}. \tag{3.55}$$

Proof. Follows immediately from (3.52) and Lemma 3.4.2, taking into account that $f^{(2)}(0) = 0$ for our choices of $f^{(2)}$. ☐

Proposition 3.4.9. If $R \ge \mu(\Phi_{\mathbf{w}})$ then $||\mathbf{x}||_\infty \le R \Rightarrow |\Phi_{\mathbf{w}}(\mathbf{x})| \le R$.

Proof. Follows immediately from Proposition 3.4.8. ☐

Proposition 3.4.10. If $R \ge \mu(\Phi_{\mathbf{w}})$ then $||\mathbf{x}||_\infty \le R \Rightarrow ||\boldsymbol{\Phi}_{\mathbf{w}}^{[m]}(\mathbf{x})||_\infty \le R$, $\forall m \ge 0$. In other words, $\boldsymbol{\Phi}_{\mathbf{w}}^{[m]}(\bar{U}) \subset \bar{U}$, $\forall m \ge 0$, where $\bar{U} = \bar{B}_\infty(0, R)$.

Proof. Follows immediately from Propositions 3.4.7 and 3.4.8. ☐

Proposition 3.4.11. $\mathbf{shr}^{[n_0]}(\mathbf{x}) = \mathbf{0} \in \mathbf{R}^{n_0}$, $\forall \mathbf{x} \in \mathbf{R}^{n_0}$.

Proof. Follows immediately from Definition 3.4.3. ☐

Proposition 3.4.12. $\boldsymbol{\Phi}_{\mathbf{w}}^{[m]}(\mathbf{x}) = \sum_{h=1}^{n_0} \Phi_{\mathbf{w}} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-h]}(\mathbf{x}) \right\} \mathbf{e}_h$, $\forall m \ge n_0$.

Proof. By Definitions 3.4.4 and 3.4.3, we have:

$$\begin{aligned} \boldsymbol{\Phi}_{\mathbf{w}}^{[m]}(\mathbf{x}) &= \boldsymbol{\Phi}_{\mathbf{w}} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-1]}(\mathbf{x}) \right\} = \Phi_{\mathbf{w}} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-1]}(\mathbf{x}) \right\} \mathbf{e}_1 + \mathbf{shr} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-1]}(\mathbf{x}) \right\} = \\ &= \Phi_{\mathbf{w}} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-1]}(\mathbf{x}) \right\} \mathbf{e}_1 + \mathbf{shr} \left\{ \Phi_{\mathbf{w}} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-2]}(\mathbf{x}) \right\} \mathbf{e}_1 + \mathbf{shr} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-2]}(\mathbf{x}) \right\} \right\} = \\ &= \Phi_{\mathbf{w}} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-1]}(\mathbf{x}) \right\} \mathbf{e}_1 + \Phi_{\mathbf{w}} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-2]}(\mathbf{x}) \right\} \mathbf{e}_2 + \mathbf{shr}^{[2]} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-2]}(\mathbf{x}) \right\}, \text{ etc.} \end{aligned} \tag{3.56}$$

In general, for $1 \le k \le \min\{m, n_0\}$, we have:

$$\boldsymbol{\Phi}_{\mathbf{w}}^{[m]}(\mathbf{x}) = \sum_{h=1}^{k} \Phi_{\mathbf{w}} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-h]}(\mathbf{x}) \right\} \mathbf{e}_h + \mathbf{shr}^{[k]} \left\{ \boldsymbol{\Phi}_{\mathbf{w}}^{[m-k]}(\mathbf{x}) \right\}. \tag{3.57}$$

The proof is concluded by setting $k = n_0$ for $m \ge n_0$ and using Proposition 3.4.11. ☐

Proposition 3.4.13. If $R \ge \mu(\Phi_{\mathbf{w}})$, then $||\mathbf{x}||_\infty \le R \Rightarrow ||\boldsymbol{\Phi}_{\mathbf{w}}^{[m]}(\mathbf{x})||_\infty < \mu(\Phi_{\mathbf{w}}) \le R$, $\forall m \ge n_0$. In particular, $||\mathbf{x}||_\infty \le R \Rightarrow ||\boldsymbol{\Phi}_{\mathbf{w}}^{[jn_0]}(\mathbf{x})||_\infty < \mu(\Phi_{\mathbf{w}}) \le R$, $\forall j \ge 1$.

Proof. By Proposition 3.4.12, for $m \geq n_0$, we have:

$$\|\mathbf{\Phi}_{\mathbf{w}}^{[m]}(\mathbf{x})\|_{\infty} = \max_{1 \leq h \leq n_0} \left| \Phi_{\mathbf{w}} \left\{ \mathbf{\Phi}_{\mathbf{w}}^{[m-h]}(\mathbf{x}) \right\} \right|. \tag{3.58}$$

Since $m - h \geq m - n_0 \geq 0$ for $1 \leq h \leq n_0$, by Proposition 3.4.10, it follows that

$$\left\| \mathbf{\Phi}_{\mathbf{w}}^{[m-h]}(\mathbf{x}) \right\|_{\infty} \leq R, \forall h, \text{ hence } \max_{1 \leq h \leq n_0} \left| \Phi_{\mathbf{w}} \left\{ \mathbf{\Phi}_{\mathbf{w}}^{[m-h]}(\mathbf{x}) \right\} \right| < \mu(\Phi_{\mathbf{w}}), \tag{3.59}$$

according to Proposition 3.4.8. □

Definition 3.4.9. The sequence $\{\mathbf{y}(t); t \geq t_1\} \subset \mathbf{R}^{n_0}$ is said to be an *orbit* of $\mathbf{\Phi} : \mathbf{R}^{n_0} \to \mathbf{R}^{n_0}$ if $\mathbf{y}(t) = \mathbf{\Phi}(\mathbf{y}(t-1)), \quad \forall t \geq t_1 + 1$ [22, 148].

Proposition 3.4.14. (3.21) and (3.22) are equivalent to $\hat{\mathbf{y}}_{\mathbf{w}}(t) = \mathbf{\Phi}_{\mathbf{w}}(\hat{\mathbf{y}}_{\mathbf{w}}(t-1))$, $\forall t \geq t_1 + 1$, $\hat{\mathbf{y}}_{\mathbf{w}}(t_1) \overset{\text{def}}{=} \mathbf{s}(t_1)$. In other words, the sequence $\{\hat{\mathbf{y}}_{\mathbf{w}}(t); t \geq t_1\} \subset \mathbf{R}^{n_0}$ generated by MSP is an orbit of $\mathbf{\Phi}_{\mathbf{w}}$ [148].

Proof. Follows immediately from Definition 3.4.4. □

Notation 3.4.7. For any set U in a topological space X, denote by $\bar{U}$ the closure of U (the smallest closed set that includes U).

Definition 3.4.10. Let X be a compact space and $\mathbf{\Phi} : X \to X$ a continuous mapping. A compact set $C \subset X$ is an *attractor* for $\mathbf{\Phi}$, if there exists an open set $U \supset C$, such that $\mathbf{\Phi}(\bar{U}) \subset U$ and $C = \bigcap_{j \geq 0} \mathbf{\Phi}^{[j]}(U)$ [22].

Proposition 3.4.15. Let X be a compact space and $\mathbf{\Phi} : X \to X$ a continuous mapping. If $C = \bigcap_{j \geq 0} \mathbf{\Phi}^{[j]}(U)$ is an attractor for $\mathbf{\Phi}$ (U open, $U \supset C$, $\mathbf{\Phi}(\bar{U}) \subset U$), then the forward orbit of any point $\mathbf{x} \in U$ converges to C [22]: $\forall$ open $V \supset C \; \exists j_V \; \forall j \geq j_V \; \mathbf{\Phi}^{[j]}(\mathbf{x}) \in V$.

Proof. Provided in [22]. □

Definition 3.4.11. Let X be a compact space and $\mathbf{\Phi} : X \to X$ a continuous mapping. An open set $U \subset X$, with $\bar{U}$ compact, is called a *trapping region* for $\mathbf{\Phi}$ if $\mathbf{\Phi}(\bar{U}) \subset U$ [22].

Proposition 3.4.16. Let X be a compact space and $\mathbf{\Phi} : X \to X$ a continuous function. If U is a trapping region for $\mathbf{\Phi}$, then the compact set $C \overset{\text{def}}{=} \bigcap_{j \geq 0} \mathbf{\Phi}^{[j]}(U)$ is an attractor for $\mathbf{\Phi}$ [22].

Theorem 3.4.1. Consider the compact space $X = \bar{B}_{\infty}(0, R+1)$, where $R \geq \mu(\Phi_{\mathbf{w}})$, and the continuous mapping $\mathbf{\Phi}_{\mathbf{w}}^{[n_0]} : X \to X$. The set $U = B_{\infty}(0, R) \subset X$ is a trapping region for $\mathbf{\Phi}_{\mathbf{w}}^{[n_0]}$. In consequence, $C_{\mathbf{w}}^{[n_0]} \overset{\text{def}}{=} \bigcap_{j \geq 0} \mathbf{\Phi}_{\mathbf{w}}^{[jn_0]}(U)$ is an attractor for $\mathbf{\Phi}_{\mathbf{w}}^{[n_0]}$.

Proof. Follows immediately from Proposition 3.4.13. □

Definition 3.4.12. Define the ω-limit set of $\mathbf{s}(t_1)$ (denoted by $\omega(\mathbf{s}(t_1))$) the set of all $\bar{\mathbf{s}} \in \mathbf{R}^{n_o}$ for which there exists a sequence $\{\bar{t}_m; m \geq 0\}$ ($\bar{t}_m \to \infty$ for $m \to \infty$) such that $\lim_{m\to\infty} \|\mathbf{s}(\bar{t}_m) - \bar{\mathbf{s}}\|_\infty = 0$ [22].

Definition 3.4.13. Define the ω-limit set of $\mathbf{s}(t_1)$ with respect to $\mathbf{\Phi_w}$ (denoted by $\omega_{\mathbf{w}}(\mathbf{s}(t_1))$) the set of all $\hat{\mathbf{s}} \in \mathbf{R}^{n_o}$ for which there exists a sequence $\{\hat{t}_m; m \geq 0\}$ ($\hat{t}_m \to \infty$ for $m \to \infty$) such that $\lim_{m\to\infty} \|\mathbf{\Phi}_{\mathbf{w}}^{[\hat{t}_m]}(\mathbf{s}(t_1)) - \hat{\mathbf{s}}\|_\infty = 0$.

Proposition 3.4.17. Let X be a topological space, $\mathbf{\Phi} : X \to X$ a continuous mapping, and $\mathbf{x} \in X$. Then $\omega_{\mathbf{\Phi}}(\mathbf{x})$ (the ω-limit set of $\mathbf{x}$ with respect to $\mathbf{\Phi}$) is closed and $\mathbf{\Phi}$-invariant [22]. If X is compact, then $\omega_{\mathbf{\Phi}}(\mathbf{x})$ is nonempty.

Proof. See [22]. If X is compact, then the sequence $\{\mathbf{\Phi}^{[j]}(\mathbf{x}); j \geq 0\} \subset X$ admits a convergent subsequence, and thus $\omega_{\mathbf{\Phi}}(\mathbf{x}) \neq \emptyset$. □

Definition 3.4.14. Define the distance between the point $\mathbf{x} \in \mathbf{R}^{n_o}$ and the closed set $F \subset \mathbf{R}^{n_o}$ by $d_\infty(\mathbf{x}, F) \overset{\text{def}}{=} \inf\{\|\mathbf{x} - \mathbf{y}\|_\infty; \mathbf{y} \in F\}$.

Proposition 3.4.18. $d_\infty(\mathbf{x}, C) = \sup\{d_\infty(\mathbf{x}, \bar{V}); V \text{ open}, V \supset C\}$, $\forall \mathbf{x} \in \mathbf{R}^{n_o}$, $\forall C$ compact.

Proof. For any open $V \supset C$, we have $\bar{V} \supset V \supset C$ and therefore $d_\infty(\mathbf{x}, C) = \inf\{\|\mathbf{x} - \mathbf{y}\|_\infty; \mathbf{y} \in C\} \geq \inf\{\|\mathbf{x} - \mathbf{y}\|_\infty; \mathbf{y} \in \bar{V}\} = d_\infty(\mathbf{x}, \bar{V})$. Hence $d_\infty(\mathbf{x}, C) \geq \sup\{d_\infty(\mathbf{x}, \bar{V}); V \text{ open}, V \supset C\}$. Now, for every $\epsilon > 0$, we have $C \subset \bigcup_{\mathbf{y} \in C} B_\infty(\mathbf{y}, \epsilon)$. Since C is compact, there is a finite subcovering of C, i.e., $\exists\{\mathbf{y}_k; 1 \leq k \leq N\} \subset C$ such that $C \subset \bigcup_{k=1}^{N} B_\infty(\mathbf{y}_k, \epsilon) = V_\epsilon$. The set V_ϵ is open, with $\bar{V}_\epsilon = \bigcup_{k=1}^{N} \bar{B}_\infty(\mathbf{y}_k, \epsilon)$ compact. For every $\mathbf{z} \in \bar{V}_\epsilon$ $\exists k$, $1 \leq k \leq N$, such that $\mathbf{z} \in \bar{B}_\infty(\mathbf{y}_k, \epsilon)$, i.e., $\|\mathbf{z} - \mathbf{y}_k\|_\infty \leq \epsilon$. Then, by the triangle inequality, we have: $d_\infty(\mathbf{x}, C) \leq \|\mathbf{x} - \mathbf{y}_k\|_\infty \leq \|\mathbf{x} - \mathbf{z}\|_\infty + \|\mathbf{z} - \mathbf{y}_k\|_\infty \leq \|\mathbf{x} - \mathbf{z}\|_\infty + \epsilon$. By taking the infimum over $\mathbf{z} \in \bar{V}_\epsilon$, we obtain $d_\infty(\mathbf{x}, C) \leq d_\infty(\mathbf{x}, \bar{V}_\epsilon) + \epsilon \leq \sup\{d_\infty(\mathbf{x}, \bar{V}); V \text{ open}, V \supset C\} + \epsilon$. Since this formula holds for any $\epsilon > 0$, it follows that $d_\infty(\mathbf{x}, C) \leq \sup\{d_\infty(\mathbf{x}, \bar{V}); V \text{ open}, V \supset C\}$, and the proof is complete. □

Proposition 3.4.19. For $N > 0$, consider the sequence $\{\mathbf{X}_m; m \geq 0\}$ and its subsequences $\{\mathbf{X}_{h-1+jN}; j \geq 0\}$, $1 \leq h \leq N$. Let $\{\mathbf{X}_{m_k}; k \geq 0\}$ be a subsequence of $\{\mathbf{X}_m; m \geq 0\}$. In other words, $\{m_k; k \geq 0\} \subset \mathbf{N}$, $m_k \to \infty$ for $k \to \infty$. Then $\exists \bar{h}$, $1 \leq \bar{h} \leq N$, $\exists\{\bar{j}_n; n \geq 0\}$ such that $\{\mathbf{X}_{\bar{h}-1+\bar{j}_n N}; n \geq 0\}$ is a subsequence of $\{\mathbf{X}_{m_k}; k \geq 0\}$.

Proof. $\forall k \geq 0$, define $h_k, j_k \in \mathbf{N}$ (uniquely determined) by $m_k = h_k - 1 + j_k N$, $1 \leq h_k \leq N$. In other words, $j_k = m_k \text{ div } N$ (the integer division quotient), and $h_k - 1 = m_k \bmod N$ (the residual). Now, $\{h_k; k \geq 0\} \subset \mathbf{N}$ is a bounded sequence, hence (by Cesaro's lemma) it has a convergent (that is, constant) subsequence $h_{k_n} = \bar{h}$, $\forall n \geq 0$, $1 \leq \bar{h} \leq N$. Note that $\{k_n; n \geq 0\} \subset \mathbf{N}$, $k_n \to \infty$ for $n \to \infty$. Hence $m_{k_n} = \bar{h} - 1 + j_{k_n} N$, and $j_{k_n} \to \infty$ as $n \to \infty$ because $n \to \infty \Rightarrow k_n \to \infty \Rightarrow m_{k_n} \to \infty$. Denote j_{k_n} by $\bar{j}_n$ and the proof is complete. □

Theorem 3.4.2. Let R be sufficiently large such that

$$R \geq \mu(\Phi_{\mathbf{w}}), \quad \Phi_{\mathbf{w}}^{[h-1]}(\mathbf{s}(t_1)) \in U = B_\infty(0, R), \quad 1 \leq h \leq n_0. \tag{3.60}$$

Then $\omega_{\mathbf{w}}(\mathbf{s}(t_1)) \subset C_{\mathbf{w}}^{[n_0]}$.

Proof. By (3.60) and by Theorem 3.4.1 and Proposition 3.4.15, all sequences

$$\left\{\Phi^{[jn_0]}\left(\Phi_{\mathbf{w}}^{[h-1]}(\mathbf{s}(t_1))\right) = \Phi_{\mathbf{w}}^{[h-1+jn_0]}(\mathbf{s}(t_1)); j \geq 0\right\}, \quad 1 \leq h \leq n_0 \tag{3.61}$$

converge to $C_{\mathbf{w}}^{[n_0]}$. By choosing an arbitrary $\hat{\mathbf{s}} \in \omega_{\mathbf{w}}(\mathbf{s}(t_1))$, there exists a subsequence $\{\hat{t}_m; m \geq 0\}$ ($\hat{t}_m \to \infty$ for $m \to \infty$) such that

$$\lim_{m\to\infty} \left\|\Phi_{\mathbf{w}}^{[\hat{t}_m]}(\mathbf{s}(t_1)) - \hat{\mathbf{s}}\right\|_\infty = 0. \tag{3.62}$$

By proposition 3.4.19 (with $N = n_0$), $\exists \hat{h}$, $1 \leq \hat{h} \leq n_0$, $\exists\{\hat{\jmath}_m; m \geq 0\}$ ($\hat{\jmath}_m \to \infty$ for $m \to \infty$) such that $\left\{\Phi_{\mathbf{w}}^{[\hat{h}-1+\hat{\jmath}_m n_0]}(\mathbf{s}(t_1)); m \geq 0\right\}$ is a subsequence of $\left\{\Phi_{\mathbf{w}}^{[\hat{t}_m]}(\mathbf{s}(t_1)); m \geq 0\right\}$. Hence:

$$\lim_{m\to\infty} \left\|\Phi_{\mathbf{w}}^{[\hat{h}-1+\hat{\jmath}_m n_0]}(\mathbf{s}(t_1)) - \hat{\mathbf{s}}\right\|_\infty = 0. \tag{3.63}$$

Fix a small $\theta > 0$. Then $\exists m_\theta$ such that

$$\left\|\Phi_{\mathbf{w}}^{[\hat{h}-1+\hat{\jmath}_m n_0]}(\mathbf{s}(t_1)) - \hat{\mathbf{s}}\right\|_\infty \leq \theta, \ \forall m \geq m_\theta. \tag{3.64}$$

Let V be an open set that includes $C_{\mathbf{w}}^{[n_0]}$. Then, by Theorem 3.4.1 and Proposition 3.4.15, $\exists m_V \geq m_\theta$ such that $\Phi_{\mathbf{w}}^{[\hat{h}-1+\hat{\jmath}_m n_0]}(\mathbf{s}(t_1)) \in V \subset \bar{V}$, $\forall m \geq m_V$. Hence

$$d_\infty(\hat{\mathbf{s}}, \bar{V}) \leq \left\|\Phi_{\mathbf{w}}^{[\hat{h}-1+\hat{\jmath}_m n_0]}(\mathbf{s}(t_1)) - \hat{\mathbf{s}}\right\|_\infty \leq \theta, \quad \forall m \geq m_V. \tag{3.65}$$

Since $d_\infty(\hat{\mathbf{s}}, \bar{V}) \leq \theta$ holds for any open $V \supset C_{\mathbf{w}}^{[n_0]}$, by Proposition 3.4.18, it follows that

$$d_\infty\left(\hat{\mathbf{s}}, C_{\mathbf{w}}^{[n_0]}\right) = \sup\left\{d_\infty(\hat{\mathbf{s}}, \bar{V}); V \text{ open}, V \supset C_{\mathbf{w}}^{[n_0]}\right\} \leq \theta. \tag{3.66}$$

Hence $d_\infty\left(\hat{\mathbf{s}}, C_{\mathbf{w}}^{[n_0]}\right) \leq \theta$ for any $\theta > 0$. For $\theta \to 0$, we obtain $d_\infty\left(\hat{\mathbf{s}}, C_{\mathbf{w}}^{[n_0]}\right) = 0$, or equivalently, $\hat{\mathbf{s}} \in C_{\mathbf{w}}^{[n_0]}$. Since $\hat{\mathbf{s}} \in \omega_{\mathbf{w}}(\mathbf{s}(t_1))$ was arbitrary, it follows that $\omega_{\mathbf{w}}(\mathbf{s}(t_1)) \subset C_{\mathbf{w}}^{[n_0]}$, and the proof is complete. ☐

Definition 3.4.15. The sequence $\{\mathbf{x}(t); t \geq t_1\} \subset \mathbf{R}^{n_0}$ is said to be a *b-pseudoorbit* of $\Phi : \mathbf{R}^{n_0} \to \mathbf{R}^{n_0}$ (for some $b > 0$), if $||\mathbf{x}(t) - \Phi(\mathbf{x}(t-1))||_\infty \leq b, \quad \forall t \geq t_1 + 1$ [22, 148].

Lemma 3.4.3. $\mathbf{x}(t) - \Phi_{\mathbf{w}}(\mathbf{x}(t-1)) = [x(t) - \Phi_{\mathbf{w}}(\mathbf{x}(t-1))]\mathbf{e}_1$, $\forall t \geq n_0$, $\forall\{\mathbf{x}(t); t \geq n_0\} \subset \mathbf{R}^{n_0}$.

Proof. Follows immediately from Definition 3.4.4. ☐

Definition 3.4.16. Given the sequence $\{\mathbf{s}(t); t \geq t_1\} \subset \mathbf{R}^{n_o}$ and the mapping $\mathbf{\Phi_w} : \mathbf{R}^{n_o} \to \mathbf{R}^{n_o}$, $\tilde{b}^{\text{tr}}_{\mathbf{w}} \overset{\text{def}}{=} \max\{||\mathbf{s}(t) - \mathbf{\Phi_w}(\mathbf{s}(t-1))||_\infty; t_0+1 \leq t \leq t_1\}$, $\tilde{b}^{\text{ts}}_{\mathbf{w}} \overset{\text{def}}{=} \sup\{||\mathbf{s}(t) - \mathbf{\Phi_w}(\mathbf{s}(t-1))||_\infty; t \geq t_1+1\}$, or equivalently: $\tilde{b}^{\text{tr}}_{\mathbf{w}} \overset{\text{def}}{=} \max\{|\tilde{e}_{\mathbf{w}}(t)|; t_0 + 1 \leq t \leq t_1\}$, $\tilde{b}^{\text{ts}}_{\mathbf{w}} \overset{\text{def}}{=} \sup\{|\tilde{e}_{\mathbf{w}}(t)|; t \geq t_1 + 1\}$.

Remark 3.4.1. There is no guarantee that $\tilde{b}^{\text{ts}}_{\mathbf{w}}$ is finite. This issue will be clarified in the following.

Definition 3.4.17. Given $\{\mathbf{s}(t); t \geq t_1\} \subset \mathbf{R}^{n_o}$: $\mathbf{s}^{++}(t) \overset{\text{def}}{=} [s(t), s(t-1), \ldots, s(t-n_0)]^T \in \mathbf{R}^{n_o}$,

$$\Delta \overset{\text{def}}{=} \sup_{t \geq t_1+1} \min_{t_0+1 \leq \tau \leq t_1} \left|\left|\mathbf{s}^{++}(t) - \mathbf{s}^{++}(\tau)\right|\right|_\infty \tag{3.67}$$

Lemma 3.4.4. $\left|\left|\mathbf{s}^{++}(t) - \mathbf{s}^{++}(\tau)\right|\right|_\infty = \max\left\{||\mathbf{s}(t) - \mathbf{s}(\tau)||_\infty, ||\mathbf{s}(t-1) - \mathbf{s}(\tau-1)||_\infty\right\}$.

Proof. Obvious. ☐

Proposition 3.4.20. $\tilde{b}^{\text{ts}}_{\mathbf{w}} \leq \tilde{b}^{\text{tr}}_{\mathbf{w}} + (1 + \lambda(\mathbf{\Phi_w}))\Delta$.

Proof. For any $t \geq t_1 + 1$, let $\tau_t = \arg \min_{t_0+1 \leq \tau \leq t_1} \left|\left|\mathbf{s}^{++}(t) - \mathbf{s}^{++}(\tau)\right|\right|_\infty$. Then, by Proposition 3.4.6, Definitions 3.4.16, 3.4.17, and Lemma 3.4.4, we have: $||\mathbf{s}(t) - \mathbf{\Phi_w}(\mathbf{s}(t-1))||_\infty \leq ||\mathbf{s}(t) - \mathbf{s}(\tau_t)||_\infty + ||\mathbf{s}(\tau_t) - \mathbf{\Phi_w}(\mathbf{s}(\tau_t - 1))||_\infty + ||\mathbf{\Phi_w}(\mathbf{s}(\tau_t - 1)) - \mathbf{\Phi_w}(\mathbf{s}(t-1))||_\infty \leq ||\mathbf{s}(t) - \mathbf{s}(\tau_t)||_\infty + \tilde{b}^{\text{tr}}_{\mathbf{w}} + \lambda(\mathbf{\Phi_w})||\mathbf{s}(t-1) - \mathbf{s}(\tau_t - 1)||_\infty \leq \tilde{b}^{\text{tr}}_{\mathbf{w}} + (1 + \lambda(\mathbf{\Phi_w})) \max\left\{||\mathbf{s}(t) - \mathbf{s}(\tau_t)||_\infty, ||\mathbf{s}(t-1) - \mathbf{s}(\tau_t - 1)||_\infty\right\} = \tilde{b}^{\text{tr}}_{\mathbf{w}} + (1 + \lambda(\mathbf{\Phi_w}))\left|\left|\mathbf{s}^{++}(t) - \mathbf{s}^{++}(\tau_t)\right|\right|_\infty \leq \tilde{b}^{\text{tr}}_{\mathbf{w}} + (1 + \lambda(\mathbf{\Phi_w}))\Delta$. ☐

Proposition 3.4.21. If $\Delta < \infty$ (or equivalently, if the trajectory $\{\mathbf{s}(t); t \geq t_1 + 1\}$ is bounded) then $\tilde{b}^{\text{ts}}_{\mathbf{w}} < \infty$ and $\{\mathbf{s}(t); t \geq t_1 + 1\}$ is a $\tilde{b}^{\text{ts}}_{\mathbf{w}}$-pseudoorbit of $\mathbf{\Phi_w}$.

Proof. Follows immediately from Definitions 3.4.15, 3.4.16 and Proposition 3.4.20. ☐

Remark 3.4.2. In all our experiments, it has been observed that, once the ANN has been trained (and $\tilde{b}^{\text{tr}}_{\mathbf{w}}$ is very small), then $\tilde{b}^{\text{ts}}_{\mathbf{w}}$ is also very small. In other words, the OSP is very accurate throughout the entire trajectory.

Definition 3.4.18. Define $\bar{e}^{\text{tr}}_{\mathbf{w}} \overset{\text{def}}{=} \max\{||\mathbf{\Phi_w}(\mathbf{s}(t-1)) - \bar{\mathbf{\Phi}}(\mathbf{s}(t-1))||_\infty; t_0 + 1 \leq t \leq t_1\}$, and $\bar{e}^{\text{ts}}_{\mathbf{w}} \overset{\text{def}}{=} \max\{||\mathbf{\Phi_w}(\mathbf{s}(t-1)) - \bar{\mathbf{\Phi}}(\mathbf{s}(t-1))||_\infty; t \geq t_1 + 1\}$, where $\bar{\Phi}$ is the mapping in (3.8) and $\bar{\mathbf{\Phi}}$ is the mapping given by Definition 3.4.5.

Proposition 3.4.22. $\bar{e}^{\text{tr}}_{\mathbf{w}} \leq \tilde{b}^{\text{tr}}_{\mathbf{w}} + \bar{e}_\infty$.

Proof. By Definitions 3.4.16 and 3.4.18, for every t, $t_0 + 1 \leq t \leq t_1$, we have: $||\mathbf{\Phi_w}(\mathbf{s}(t-1)) - \bar{\mathbf{\Phi}}(\mathbf{s}(t-1))||_\infty \leq ||\mathbf{\Phi_w}(\mathbf{s}(t-1)) - \mathbf{s}(t)||_\infty + ||\mathbf{s}(t) - \bar{\mathbf{\Phi}}(\mathbf{s}(t-1))||_\infty \leq \tilde{b}^{\text{tr}}_{\mathbf{w}} + \bar{e}_\infty$. ☐

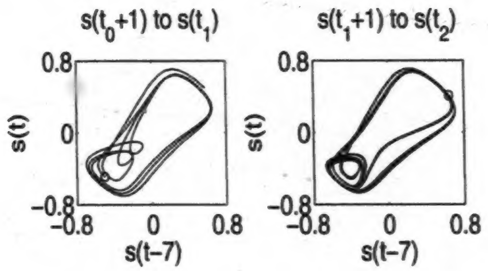

Figure 3.9: Phase plot for training set (left) and testing set (right).

Proposition 3.4.23. If $\bar{\Phi} \in C^0(\mathbf{R}^{n_o}; \mathbf{R})$, then $\forall \epsilon > 0$, $\exists \delta_{\mathbf{w}}(\epsilon) > 0\ \forall \mathbf{x} \in \bigcup_{t=t_0+1}^{t_1} \bar{B}_\infty(\mathbf{s}(t-1), \delta_{\mathbf{w}}(\epsilon))$: $||\mathbf{\Phi}_{\mathbf{w}}(\mathbf{x}) - \bar{\mathbf{\Phi}}(\mathbf{x})||_\infty \leq \tilde{b}_{\mathbf{w}}^{\mathrm{tr}} + \bar{e}_\infty + \epsilon$.

Proof. Since $\mathbf{\Phi}_{\mathbf{w}}(\cdot)$, $\bar{\mathbf{\Phi}}(\cdot)$, $||\cdot||_\infty$ are continuous, it follows that the mapping $\mathbf{x} \mapsto ||\mathbf{\Phi}_{\mathbf{w}}(\mathbf{x}) - \bar{\mathbf{\Phi}}(\mathbf{x})||_\infty$ is continuous, which proves the proposition. □

Definition 3.4.19. For a given $\epsilon > 0$, let $\bar{\delta}_{\mathbf{w}}(\epsilon)$ be the maximum $\delta_{\mathbf{w}}(\epsilon)$ satisfying Proposition 3.4.23. Let $\bar{\mathcal{E}}_{\mathbf{w}} \overset{\mathrm{def}}{=} \{\epsilon > 0; \bar{\delta}_{\mathbf{w}}(\epsilon) \geq \Delta\}$ and $\bar{\epsilon}_{\mathbf{w}} \overset{\mathrm{def}}{=} \inf \bar{\mathcal{E}}_{\mathbf{w}}$.

Proposition 3.4.24. If $\bar{\Phi} \in C^0(\mathbf{R}^{n_o}; \mathbf{R})$ and $\bar{\mathcal{E}}_{\mathbf{w}}$ is nonempty, then $\tilde{b}_{\mathbf{w}}^{\mathrm{ts}} \leq \tilde{b}_{\mathbf{w}}^{\mathrm{tr}} + 2\bar{e}_\infty + \bar{\epsilon}_{\mathbf{w}}$.

Proof. Let $\epsilon \in \bar{\mathcal{E}}_{\mathbf{w}}$, that is, $\exists \delta_{\mathbf{w}}(\epsilon) \geq \Delta$. For any $t \geq t_1+1$, let $\tau_t = \arg \min_{t_0+1 \leq \tau \leq t_1} ||\mathbf{s}^{++}(t) - \mathbf{s}^{++}(\tau)||_\infty$. Then, by Lemma 3.4.4, we have: $||\mathbf{s}(t-1) - \mathbf{s}(\tau_t - 1)||_\infty \leq ||\mathbf{s}^{++}(t) - \mathbf{s}^{++}(\tau_t)||_\infty \leq \Delta \leq \delta_{\mathbf{w}}(\epsilon)$. Hence $\mathbf{s}(t-1) \in \bar{B}_\infty(\mathbf{s}(\tau_t - 1), \delta_{\mathbf{w}}(\epsilon))$, and by Proposition 3.4.23 it follows that $||\mathbf{\Phi}_{\mathbf{w}}(\mathbf{s}(t-1)) - \bar{\mathbf{\Phi}}(\mathbf{s}(t-1))||_\infty \leq \tilde{b}_{\mathbf{w}}^{\mathrm{tr}} + \bar{e}_\infty + \epsilon$. Hence $||\mathbf{s}(t) - \mathbf{\Phi}_{\mathbf{w}}(\mathbf{s}(t-1))||_\infty \leq ||\mathbf{s}(t) - \bar{\mathbf{\Phi}}(\mathbf{s}(t-1))||_\infty + ||\bar{\mathbf{\Phi}}(\mathbf{s}(t-1)) - \mathbf{\Phi}_{\mathbf{w}}(\mathbf{s}(t-1))||_\infty \leq \bar{e}_\infty + \tilde{b}_{\mathbf{w}}^{\mathrm{tr}} + \bar{e}_\infty + \epsilon$. □

Remark 3.4.3. Both Propositions 3.4.20 and 3.4.24 imply that the OSP throughout the given trajectory is accurate (i.e., $\tilde{b}_{\mathbf{w}}^{\mathrm{ts}}$ is small) if Δ is small. In particular, if Δ is small then most likely $\bar{\mathcal{E}}_{\mathbf{w}}$ will be nonempty. A small Δ means that the successive oscillation cycles stay close to each other, as in Fig.3.9. This happens, for instance, when the given trajectory converges to a limit cycle. If the ANN training set is close enough to the limit cycle of the given trajectory, then the successive oscillation cycles are actually closer and closer to each other, and therefore Δ is small.

Definition 3.4.20. We say that $\mathbf{\Phi}$ has the *shadowing property*, if $\forall \epsilon > 0$, $\exists b(\epsilon) > 0\ \forall \{\mathbf{x}(t); t \geq t_1\}$, a $b(\epsilon)$-pseudoorbit of $\mathbf{\Phi}$, $\exists \{\mathbf{z}_{\epsilon,\mathbf{x}(\cdot)}(t); t \geq t_1\}$ an orbit of $\mathbf{\Phi}$ such that $||\mathbf{z}_{\epsilon,\mathbf{x}(\cdot)}(t) - \mathbf{x}(t)||_\infty \leq \epsilon$ $\forall t \geq t_1$. In other words, the corrupted trajectories are "shadowed" by the real trajectories [148].

Proposition 3.4.25. The mapping $\boldsymbol{\Phi}_{\mathbf{w}}$ has the shadowing property [148]. In other words, $\forall \epsilon > 0$, $\exists b_{\mathbf{w}}(\epsilon) > 0$ $\forall \{\mathbf{x}(t); t \geq t_1\}$ a $b_{\mathbf{w}}(\epsilon)$-pseudoorbit of $\boldsymbol{\Phi}_{\mathbf{w}}$, $\exists \{\mathbf{z}_{\epsilon,\mathbf{x}(\cdot)}(t); t \geq t_1\}$ an orbit of $\boldsymbol{\Phi}_{\mathbf{w}}$ such that $||\mathbf{z}_{\epsilon,\mathbf{x}(\cdot)}(t) - \mathbf{x}(t)||_\infty \leq \epsilon$ $\forall t \geq t_1$.

Remark 3.4.4. According to Lin et.al. [148], it has been proven that, for all practical purposes, analog ANNs (i.e., ANNs whose inputs and outputs can each take values in an interval) do have the shadowing property.

Definition 3.4.21. For every $\epsilon > 0$, let $B_{\mathbf{w}}(\epsilon)$ be the set of all $b_{\mathbf{w}}(\epsilon)$ in Proposition 3.4.25, and let $\tilde{b}_{\mathbf{w}}(\epsilon) \overset{\text{def}}{=} \sup B_{\mathbf{w}}(\epsilon)$.

Remark 3.4.5. In general, the mapping $\epsilon \mapsto \tilde{b}_{\mathbf{w}}(\epsilon)$ is expected to be increasing [148].

Definition 3.4.22. Let $\tilde{\mathcal{E}}_{\mathbf{w}} \overset{\text{def}}{=} \left\{\epsilon > 0; \tilde{b}_{\mathbf{w}}(\epsilon) \geq \tilde{b}^{\mathrm{ts}}_{\mathbf{w}}\right\}$ and $\tilde{\epsilon}_{\mathbf{w}} \overset{\text{def}}{=} \inf \tilde{\mathcal{E}}_{\mathbf{w}}$.

Proposition 3.4.26. $\tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}}) \in \tilde{\mathcal{E}}_{\mathbf{w}}$, $\tilde{b}^{\mathrm{ts}}_{\mathbf{w}} \in B_{\mathbf{w}}\left(\tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}})\right)$. Hence the set $\tilde{\mathcal{E}}_{\mathbf{w}}$ is nonempty and $\tilde{\epsilon}_{\mathbf{w}} \leq \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}})$. Also:

$$\tilde{\epsilon}_{\mathbf{w}} = \inf \tilde{\mathcal{E}}_{\mathbf{w}} = \inf \left\{\tilde{\mathcal{E}}_{\mathbf{w}} \cap \left[0, \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}})\right]\right\}. \tag{3.68}$$

Proof. We show that $\epsilon = \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}})$ and $b_{\mathbf{w}}(\epsilon) = \tilde{b}^{\mathrm{ts}}_{\mathbf{w}}$ satisfy the property given in Proposition 3.4.25. For any $b_{\mathbf{w}}(\epsilon)$-pseudoorbit $\{\mathbf{x}(t); t \geq t_1\}$ of $\boldsymbol{\Phi}_{\mathbf{w}}$, choose the orbit $\{\mathbf{y}(t); t \geq t_1\}$ of $\boldsymbol{\Phi}_{\mathbf{w}}$ for which $\mathbf{y}(t_1) \overset{\text{def}}{=} \mathbf{x}(t_1)$. We show that $||\mathbf{y}(t) - \mathbf{x}(t)||_\infty \leq \epsilon$ $\forall t \geq t_1$. Note that, by the choice of $\{\mathbf{y}(t); t \geq t_1\}$, we have $||\mathbf{y}(t_1+1) - \mathbf{x}(t_1+1)||_\infty = |y(t_1+1) - x(t_1+1)|$, $||\mathbf{y}(t_1+2) - \mathbf{x}(t_1+2)||_\infty = \max\{|y(t_1+2) - x(t_1+2)|, |y(t_1+1) - x(t_1+1)|\}$, etc. In general, $\forall t \geq t_1 + 1$:

$$||\mathbf{y}(t) - \mathbf{x}(t)||_\infty = \max\{|y(t-j+1) - x(t-j+1)|; 1 \leq j \leq \min\{t - t_1, n_0\}\} \tag{3.69}$$

where all $t - j + 1 \geq t_1 + 1$. On the other hand, $\forall t \geq t_1 + 1$: $|y(t) - x(t)| \leq |y(t) - \Phi_{\mathbf{w}}(\mathbf{x}(t-1))| + |\Phi_{\mathbf{w}}(\mathbf{x}(t-1)) - x(t)| \leq |y(t)| + |\Phi_{\mathbf{w}}(\mathbf{x}(t-1))| + b_{\mathbf{w}}(\epsilon) = |\Phi_{\mathbf{w}}(\mathbf{y}(t-1))| + |\Phi_{\mathbf{w}}(\mathbf{x}(t-1))| + \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} \leq 2\mu(\Phi_{\mathbf{w}}) + \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} = \epsilon$. Hence $|y(t-j+1) - x(t-j+1)| \leq \epsilon$, $1 \leq j \leq \min\{t - t_1, n_0\}$, and therefore $||\mathbf{y}(t) - \mathbf{x}(t)||_\infty \leq \epsilon$, $\forall t \geq t_1 + 1$. Together with $||\mathbf{y}(t_1) - \mathbf{x}(t_1)||_\infty = 0$, this completes the proof. □

Theorem 3.4.3. Let R be sufficiently large such that, for $1 \leq h \leq n_0$:

$$R \geq \mu(\Phi_{\mathbf{w}}), \quad \boldsymbol{\Phi}^{[h-1]}_{\mathbf{w}}(\mathbf{s}(t_1)) \in U, \quad \bar{B}_\infty\left(\mathbf{s}(t_1 + h - 1), \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}})\right) \subset U \tag{3.70}$$

where $U = B_\infty(0, R)$. Then

$$d_\infty\left(\bar{\mathbf{s}}, C^{[n_0]}_{\mathbf{w}}\right) \leq \tilde{\epsilon}_{\mathbf{w}}, \quad \forall \bar{\mathbf{s}} \in \omega(\mathbf{s}(t_1)). \tag{3.71}$$

Proof. Let $\epsilon \in \tilde{\mathcal{E}}_{\mathbf{w}}$, $\epsilon \leq \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}})$. By Definition 3.4.22, $\tilde{b}_{\mathbf{w}}(\epsilon) \geq \tilde{b}^{\mathrm{ts}}_{\mathbf{w}}$, and by Proposition 3.4.21, it follows that $\{\mathbf{s}(t); t \geq t_1\}$ is a $\tilde{b}_{\mathbf{w}}(\epsilon)$-pseudoorbit of $\mathbf{\Phi}_{\mathbf{w}}$. By Proposition 3.4.25, $\exists\{\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t); t \geq t_1\}$ an orbit of $\mathbf{\Phi}_{\mathbf{w}}$ such that

$$||\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t) - \mathbf{s}(t)||_\infty \leq \epsilon, \ \forall t \geq t_1. \tag{3.72}$$

In particular:

$$||\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + h - 1) - \mathbf{s}(t_1 + h - 1)||_\infty \leq \epsilon, \ 1 \leq h \leq n_0. \tag{3.73}$$

In other words, $\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + h - 1) \in \bar{B}_\infty(\mathbf{s}(t_1 + h - 1), \epsilon)$, $1 \leq h \leq n_0$. Since we have chosen $\epsilon \leq \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}})$, by (3.70) it follows that, for $1 \leq h \leq n_0$:

$$\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + h - 1) \in \bar{B}_\infty(\mathbf{s}(t_1 + h - 1), \epsilon) \subset \bar{B}_\infty\left(\mathbf{s}(t_1 + h - 1), \tilde{b}^{\mathrm{ts}}_{\mathbf{w}} + 2\mu(\Phi_{\mathbf{w}})\right) \subset U. \tag{3.74}$$

By writing (3.72) for $t = t_1 + h - 1 + jn_0$, $1 \leq h \leq n_0$, $j \geq 0$, we obtain:

$$||\mathbf{\Phi}^{[jn_0]}_{\mathbf{w}}(\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + h - 1)) - \mathbf{s}(t_1 + h - 1 + jn_0)||_\infty \leq \epsilon. \tag{3.75}$$

where

$$\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + h - 1) \in U, \quad 1 \leq h \leq n_0. \tag{3.76}$$

Let $\bar{\mathbf{s}} \in \omega(\mathbf{s}(t_1))$, then there exists a subsequence $\{\bar{t}_m; m \geq 0\}$ ($\bar{t}_m \to \infty$ for $m \to \infty$) such that

$$\lim_{m\to\infty} ||\mathbf{s}(\bar{t}_m) - \bar{\mathbf{s}}||_\infty = 0. \tag{3.77}$$

According to Proposition 3.4.19 (with $N = n_0$), $\exists\bar{h}$, $1 \leq \bar{h} \leq n_0$, $\exists\{\bar{\jmath}_m; m \geq 0\}$ ($\bar{\jmath}_m \to \infty$ for $m \to \infty$) such that $\{\mathbf{s}(t_1 + \bar{h} - 1 + \bar{\jmath}_m n_0); m \geq 0\}$ is a subsequence of $\{\mathbf{s}(\bar{t}_m); m \geq 0\}$. Hence, by (3.77):

$$\lim_{m\to\infty} ||\mathbf{s}(t_1 + \bar{h} - 1 + \bar{\jmath}_m n_0) - \bar{\mathbf{s}}||_\infty = 0. \tag{3.78}$$

Select a small $\theta > 0$. Then $\exists m_\theta$ such that

$$||\mathbf{s}(t_1 + \bar{h} - 1 + \bar{\jmath}_m n_0) - \bar{\mathbf{s}}||_\infty \leq \theta, \ \forall m \geq m_\theta. \tag{3.79}$$

By (3.75), we also have

$$||\mathbf{\Phi}^{[\bar{\jmath}_m n_0]}_{\mathbf{w}}(\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + \bar{h} - 1)) - \mathbf{s}(t_1 + \bar{h} - 1 + \bar{\jmath}_m n_0)||_\infty \leq \epsilon, \ \forall m \geq 0 \tag{3.80}$$

and, by (3.76), Theorem 3.4.1 and Proposition 3.4.15, it follows that all $\mathbf{\Phi}^{[n_0]}_{\mathbf{w}}$-orbits starting in $\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + \bar{h} - 1)$ converge to the attractor $C^{[n_0]}_{\mathbf{w}}$.

Let V be an open set that includes $C^{[n_0]}_{\mathbf{w}}$. Then $\exists m_V \geq m_\theta$ such that

$$\mathbf{\Phi}^{[\bar{\jmath}_m n_0]}_{\mathbf{w}}(\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + \bar{h} - 1)) \in V \subset \bar{V}, \quad \forall m \geq m_V. \tag{3.81}$$

By (3.79) and (3.80) and the triangle inequality, we obtain

$$d_\infty(\bar{\mathbf{s}}, \bar{V}) \le ||\mathbf{\Phi}_{\mathbf{w}}^{[\bar{j}_m n_o]}(\mathbf{z}_{\epsilon,\mathbf{s}(\cdot)}(t_1 + \bar{h} - 1)) - \bar{\mathbf{s}}||_\infty \le \epsilon + \theta, \ \forall m \ge m_\theta. \tag{3.82}$$

Since $d_\infty(\bar{\mathbf{s}}, \bar{V}) \le \epsilon + \theta$ holds for any open $V \supset C_{\mathbf{w}}^{[n_o]}$, by Proposition 3.4.18, it follows that

$$d_\infty\left(\bar{\mathbf{s}}, C_{\mathbf{w}}^{[n_o]}\right) = \sup\left\{d_\infty(\bar{\mathbf{s}}, \bar{V}); V \text{ open}, V \supset C_{\mathbf{w}}^{[n_o]}\right\} \le \epsilon + \theta \tag{3.83}$$

and the above inequality holds for any $\theta > 0$. Hence, we conclude that

$$d_\infty\left(\bar{\mathbf{s}}, C_{\mathbf{w}}^{[n_o]}\right) \le \epsilon, \quad \forall \bar{\mathbf{s}} \in \omega(\mathbf{s}(t_1)). \tag{3.84}$$

The above equation holds for any $\epsilon \in \tilde{\mathcal{E}}_{\mathbf{w}}$, $\epsilon \le \tilde{b}_{\mathbf{w}}^{\text{ts}} + 2\mu(\Phi_{\mathbf{w}})$. Using Proposition 3.4.26 and taking the infimum over $\epsilon \in \tilde{\mathcal{E}}_{\mathbf{w}} \cap \left[0, \tilde{b}_{\mathbf{w}}^{\text{ts}} + 2\mu(\Phi_{\mathbf{w}})\right]$, we obtain:

$$d_\infty\left(\bar{\mathbf{s}}, C_{\mathbf{w}}^{[n_o]}\right) \le \tilde{\epsilon}_{\mathbf{w}}, \quad \forall \bar{\mathbf{s}} \in \omega(\mathbf{s}(t_1)). \tag{3.85}$$

□

Definition 3.4.23. Let X be a compact space and $\mathbf{\Phi} : X \to X$ a continuous mapping. A closed, non-empty, forward $\mathbf{\Phi}$-invariant subset $Y \subset X$ is called a *minimal set* for $\mathbf{\Phi}$, if it contains no proper closed, non-empty, forward $\mathbf{\Phi}$-invariant subset [22].

Proposition 3.4.27. Let X be a compact space and $\mathbf{\Phi} : X \to X$ a continuous mapping. $Y \subset X$ is a minimal set for $\mathbf{\Phi}$ if and only if $\omega_{\mathbf{\Phi}}(y) = Y$, $\forall y \in Y$ [22].

Remark 3.4.6. A periodic orbit is a minimal set [22].

Proposition 3.4.28. Let X be a compact space and $\mathbf{\Phi} : X \to X$ a continuous mapping. Then X contains a minimal set for $\mathbf{\Phi}$ [22].

Proof. Provided in [22]. □

Proposition 3.4.29. Let $\mathbf{\Phi} : \mathbf{R}^{n_o} \to \mathbf{R}^{n_o}$ be a continuous mapping, and $U \subset X$ an open set, with $\bar{U}$ compact. Then $\overline{\mathbf{\Phi}(U)} \subset \mathbf{\Phi}(\bar{U})$.

Proof. For every $\bar{\mathbf{y}} \in \overline{\mathbf{\Phi}(U)}$, there exists a sequence $\{\bar{\mathbf{x}}_n; n \ge 0\} \subset U \subset \bar{U}$ such that $\mathbf{\Phi}(\bar{\mathbf{x}}_n) \to \bar{\mathbf{y}}$ as $n \to \infty$. Since $\bar{U}$ is compact, it follows that there is a convergent subsequence $\{\bar{\mathbf{x}}_{n_k}; k \ge 0\}$, $\bar{\mathbf{x}}_{n_k} \to \bar{\mathbf{x}}$ (for some $\bar{\mathbf{x}} \in \bar{U}$) as $k \to \infty$. Hence, by the continuity of $\mathbf{\Phi}$: $\mathbf{\Phi}(\bar{\mathbf{x}}_{n_k}) \to \mathbf{\Phi}(\bar{\mathbf{x}})$ as $k \to \infty$. On the other hand $\mathbf{\Phi}(\bar{\mathbf{x}}_{n_k}) \to \bar{\mathbf{y}}$ as $k \to \infty$. Hence $\bar{\mathbf{y}} = \mathbf{\Phi}(\bar{\mathbf{x}})$ where $\bar{\mathbf{x}} \in \bar{U}$. In conclusion, $\bar{\mathbf{y}} \in \mathbf{\Phi}(\bar{U})$. □

Lemma 3.4.5. $\overline{\bigcap_{j \ge 0} A_j} \subset \bigcap_{j \ge 0} \overline{A_j}$, for any family of sets $\{A_j; j \ge 0\}$ in $\mathbf{R}^{n_o}$.

Proof. For every $\bar{\mathbf{x}} \in \overline{\bigcap_{j\geq 0} A_j}$, there exists a sequence $\{\bar{\mathbf{x}}_n; n \geq 0\} \subset \bigcap_{j\geq 0} A_j$ such that $\bar{\mathbf{x}}_n \to \bar{\mathbf{x}}$ as $n \to \infty$. It follows that, $\forall j$, $\{\bar{\mathbf{x}}_n; n \geq 0\} \subset A_j$ and $\bar{\mathbf{x}}_n \to \bar{\mathbf{x}}$ as $n \to \infty$. Hence $\bar{\mathbf{x}} \in \overline{A_j}$, $\forall j$, which implies that $\bar{\mathbf{x}} \in \bigcap_{j\geq 0} \overline{A_j}$. □

Proposition 3.4.30. Let $X \subset \mathbf{R}^{n_o}$ be a compact space, $\mathbf{\Phi} : X \to X$ a continuous mapping, $U \subset X$ (with $\bar{U}$ compact) a trapping region for $\mathbf{\Phi}$, and $C \overset{\text{def}}{=} \bigcap_{j\geq 0} \mathbf{\Phi}^{[j]}(U)$ the corresponding attractor for $\mathbf{\Phi}$. Then $C = \bigcap_{j\geq 0} \mathbf{\Phi}^{[j]}(\bar{U})$.

Proof. Note that $C = \bar{C}$ because C is compact, and $\mathbf{\Phi}(\bar{U}) \subset U$ since U is a trapping region for $\mathbf{\Phi}$. By applying Lemma 3.4.5 and Proposition 3.4.29, we obtain:

$$\begin{aligned} C = \bar{C} = \overline{\bigcap_{j\geq 0} \mathbf{\Phi}^{[j]}(U)} \subset \bigcap_{j\geq 0} \overline{\mathbf{\Phi}^{[j]}(U)} \subset \bigcap_{j\geq 0} \mathbf{\Phi}^{[j]}(\bar{U}) \subset \bigcap_{j\geq 0} \mathbf{\Phi}^{[j+1]}(\bar{U}) = \\ = \bigcap_{j\geq 0} \mathbf{\Phi}^{[j]}\left\{\mathbf{\Phi}(\bar{U})\right\} \subset \bigcap_{j\geq 0} \mathbf{\Phi}^{[j]}(U) = C. \end{aligned} \tag{3.86}$$

It then follows that all inclusions in the above equation are actually equalities. In particular, the conclusion of the proposition follows. □

Proposition 3.4.31. $C_{\mathbf{w}}^{[n_o]}$ is forward $\mathbf{\Phi}_{\mathbf{w}}$-invariant: $\mathbf{\Phi}_{\mathbf{w}}\left\{C_{\mathbf{w}}^{[n_o]}\right\} \subset C_{\mathbf{w}}^{[n_o]}$.

Proof. By Proposition 3.4.10, $\mathbf{\Phi}_{\mathbf{w}}(\bar{U}) \subset \bar{U}$. Hence, we have:

$$\begin{aligned} \mathbf{\Phi}_{\mathbf{w}}\left\{C_{\mathbf{w}}^{[n_o]}\right\} = \mathbf{\Phi}_{\mathbf{w}}\left\{\bigcap_{j\geq 0} \mathbf{\Phi}_{\mathbf{w}}^{[jn_o]}(\bar{U})\right\} \subset \bigcap_{j\geq 0} \mathbf{\Phi}_{\mathbf{w}}\left\{\mathbf{\Phi}_{\mathbf{w}}^{[jn_o]}(\bar{U})\right\} = \\ = \bigcap_{j\geq 0} \mathbf{\Phi}_{\mathbf{w}}^{[jn_o]}\left\{\mathbf{\Phi}_{\mathbf{w}}(\bar{U})\right\} \subset \bigcap_{j\geq 0} \mathbf{\Phi}_{\mathbf{w}}^{[jn_o]}(\bar{U}) = C_{\mathbf{w}}^{[n_o]}. \end{aligned} \tag{3.87}$$

□

Proposition 3.4.32. $C_{\mathbf{w}}^{[n_o]}$ contains a minimal set for $\mathbf{\Phi}_{\mathbf{w}}$.

Proof. Apply Proposition 3.4.28 for $X = C_{\mathbf{w}}^{[n_o]}$ and $\mathbf{\Phi} = \mathbf{\Phi}_{\mathbf{w}}$, given that $C_{\mathbf{w}}^{[n_o]}$ is compact and $\mathbf{\Phi}_{\mathbf{w}}\left\{C_{\mathbf{w}}^{[n_o]}\right\} \subset C_{\mathbf{w}}^{[n_o]}$. □

Definition 3.4.24. Let X be a topological space and $\mathbf{\Phi} : X \to X$ a continuous mapping. For every $\mathbf{x} \in X$, define the *positive semiorbit* starting from $\mathbf{x}$: $\mathcal{O}^+(\mathbf{x}) \overset{\text{def}}{=} \left\{\mathbf{\Phi}^{[j]}(\mathbf{x}); j \geq 1\right\}$ [22].

Notation 3.4.8. For every $\mathbf{x} \in \mathbf{R}^{n_o}$, denote by $\mathcal{O}_{\mathbf{w}}^+(\mathbf{x})$ the positive semiorbit starting from $\mathbf{x}$ with respect to the mapping $\mathbf{\Phi}_{\mathbf{w}}$, namely $\mathcal{O}_{\mathbf{w}}^+(\mathbf{x}) \overset{\text{def}}{=} \left\{\mathbf{\Phi}_{\mathbf{w}}^{[j]}(\mathbf{x}); j \geq 1\right\}$.

Definition 3.4.25. A subset $A \subset \mathbf{N}$ is said to be *relatively dense*, if there exists $k_A > 0$ such that $\{n, n+1, \ldots, n+k_A\} \cap A \neq \emptyset$, $\forall n \in \mathbf{N}$ [22].

Definition 3.4.26. Let X be a topological space and $\mathbf{\Phi} : X \to X$. A point $\mathbf{x} \in X$ is said to be *almost periodic*, if for any neighbourhood U of $\mathbf{x}$, the set $\{j \in \mathbf{N}; \mathbf{\Phi}^{[j]}(\mathbf{x}) \in U\}$ is relatively dense [22]. In other words: $\forall U\ \exists k_U\ \forall n\ \exists j_{n,k_U} \in \{n, n+1, \ldots, n+k_U\}$: $\mathbf{\Phi}^{[j_{n,k_U}]}(\mathbf{x}) \in U$.

Proposition 3.4.33. Let X be a compact Hausdorff space, $\mathbf{\Phi} : X \to X$ a continuous mapping, and $\mathbf{x} \in X$. Then $\overline{\mathcal{O}^{+}(\mathbf{x})}$ is minimal for $\mathbf{\Phi}$ if and only if $\mathbf{x}$ is almost periodic [22].

Proof. Provided in [22]. ☐

Proposition 3.4.34. If $C_{\mathbf{w}}^{[no]} = \overline{\mathcal{O}_{\mathbf{w}}^{+}(\bar{\mathbf{x}})}$, where $\bar{\mathbf{x}} \in \mathbf{R}^{no}$ is almost periodic, then $C_{\mathbf{w}}^{[no]}$ is minimal, $C_{\mathbf{w}}^{[no]} = \omega_{\mathbf{w}}(\mathbf{s}(t_1))$, and

$$d_\infty\left(\bar{\mathbf{s}}, \omega_{\mathbf{w}}(\mathbf{s}(t_1))\right) \le \tilde{\epsilon}_{\mathbf{w}}, \quad \forall \bar{\mathbf{s}} \in \omega(\mathbf{s}(t_1)). \tag{3.88}$$

Proof. Apply Definition 3.4.23, Theorems 3.4.2, 3.4.3, and Propositions 3.4.33, 3.4.17. ☐

Remark 3.4.7. Proposition 3.4.34 states that every point in the ω-limit set of the given trajectory is situated at a distance at most $\tilde{\epsilon}_{\mathbf{w}}$ from the ω-limit set of the trajectory generated by MSP using the proposed ANN, where $\tilde{\epsilon}_{\mathbf{w}}$ is the **smallest** ϵ for which the **largest** $b_{\mathbf{w}}(\epsilon)$ (in the shadowing property of $\mathbf{\Phi}_{\mathbf{w}}$) is larger than the maximum OSP error throughout the given trajectory.

Remark 3.4.8. The mapping $\epsilon \mapsto \tilde{b}_{\mathbf{w}}(\epsilon)$ depends on the form of $\mathbf{\Phi}_{\mathbf{w}}(\mathbf{x}) = \mathbf{\Phi}(\mathbf{w}, \mathbf{x})$ (given by the ANN architecture) and on the particular value of $\mathbf{w}$ (the ANN weights). In practice it might very well happen that, in some cases, a very small (and thus, unattainable) $\tilde{b}_{\mathbf{w}}(\epsilon)$ (desired OSP accuracy throughout the trajectory) may be required for a relatively large (and thus, useless) ϵ. Estimating the mapping $\epsilon \mapsto \tilde{b}_{\mathbf{w}}(\epsilon)$ for a given $\mathbf{\Phi}_{\mathbf{w}}$ could constitute a very interesting subject for further research.

Remark 3.4.9. In most experiments, the asymptotic state of the given trajectory is a limit cycle. It has been noticed experimentally that the MSP generated by the ANN also converges to a limit cycle whose frequency is close to the frequency of the limit cycle of the given trajectory. In that case, Proposition 3.4.34 states that the limit cycle of the signal predicted by the ANN is close to the (unknown) limit cycle of the given trajectory.

Remark 3.4.10. In our experiments, the ANN training minimized the mean-squared OSP error over the training set, given by (3.14). We have also attempted to perform a version of ANN training consisting in minimizing the mean-squared MSP error over the training set:

$$\hat{E}(\mathbf{w}) \overset{\text{def}}{=} \frac{1}{t_1 - t_0} \sum_{t=t_0+1}^{t_1} \left[s(t) - \mathbf{\Phi}_{\mathbf{w}} \left\{ \mathbf{\Phi}_{\mathbf{w}}^{[t-t_0-1]}(\mathbf{s}(t_0)) \right\} \right]^2 \tag{3.89}$$

by using the *real time recurrent learning (RTRL)* algorithm [10, 91, 158, 159, 160]. The MSP accuracy in the testing phase (after training) was at most as good as the accuracy obtained by OSP-based training. However, the MSP-based training time was much longer than the OSP-based training time, due to the fact that the former training type involves much more computations than the latter. It is not clear that MSP-based training will provide a better approximation for the mapping $\bar{\Phi}$ in (3.8). Even though towards the end of training $\mathbf{\Phi}_{\mathbf{w}}^{[t-t_0-1]}(\mathbf{s}(t_0))$ become close to $\mathbf{s}(t-1)$, the differences $\hat{\mathbf{e}}_{\mathbf{w}}(t-1) \overset{\text{def}}{=} \mathbf{s}(t-1) - \mathbf{\Phi}_{\mathbf{w}}^{[t-t_0-1]}(\mathbf{s}(t_0))$ (that is, the input noise during training), for consecutive values of t, are highly correlated. In addition, a problem with the MSP-based training may be that the ANN is actually trained to perform an accurate MSP starting from the point $\mathbf{s}(t_0)$, while in the testing phase it has to predict the portion of the trajectory starting from the point $\mathbf{s}(t_1)$. This problem could be addressed by redefining the error function as

$$\hat{E}(\mathbf{w}) \overset{\text{def}}{=} \frac{1}{t_1 - t_0} \sum_{\tau=t_0+1}^{t_1} \frac{1}{t_1 - \tau + 1} \sum_{t=\tau}^{t_1} \left[s(t) - \Phi_{\mathbf{w}} \left\{ \mathbf{\Phi}_{\mathbf{w}}^{[t-\tau]}(\mathbf{s}(\tau - 1)) \right\} \right]^2 . \tag{3.90}$$

A fast training algorithm needs to be designed in that case, since the number of terms in the error function has grown from $t_1 - t_0$ to $(t_1 - t_0)(t_1 - t_0 + 1)/2$. Further research is needed in order to thoroughly investigate this approach.

Chapter 4

Case Studies

As mentioned in the previous sections, the practical implementation of ANNs for a given problem is not as straightforward as the elegant theoretical results might suggest. Cybenko [53] notes that the number of neurons necessary in most practical problems is likely astronomically high, due to the so-called curse of dimensionality. Using many neurons is computationally expensive to the point that solving the problem may become unfeasible. Hornik [96] observes that not all transfer functions that satisfy the theoretical requirements for the universal approximation will perform equally well in practical problems. Due to the complicated shape of the error surfaces, the ANN training often converges to a local minimum, which may not be a solution of the given problem. Since in MSP applications the use of global optimization algorithms (genetic, annealing) is prohibitive because of the large number of parameters (ANN weights) to be estimated, one is forced to employ point-by-point nonlinear optimization algorithms, which depend heavily on the initial guess on the weight values. How to determine good initial values of the network weights for ANN training is still an unsolved problem.

Given these practical difficulties, we are interested in finding a means of extracting maximum information from the training set using a minimum number of hidden neurons. This would reduce training time and provide better generalization capabilities and robustness to noise, as a consequence of having few degrees-of-freedom in the system. The successful ANN architectures previously proposed [238, 239, 240, 259, 260] involved either some kind of weight scaling or using a scaled sigmoidal transfer function in the output layer. In the present study we will investigate, among other things, the effect of these two architectural features on the ANN MSP performance under neuron scarcity conditions. In all, 12 LTMSP methods using ANNs will be compared, based on their performance on 8 test cases[1].

---

[1] A version of part of the material presented in this chapter has been submitted for publication in [241, 242].

4.1 Test Cases

The test signals S1, S3, S5 are obtained from numerically solving an aeroelastic model with structural nonlinearities. The governing equations for a self-excited two-degree-of-freedom airfoil oscillating in pitch and plunge can be expressed as [140, 151]:

$$\begin{aligned} &\ddot{\xi} + x_\alpha \ddot{\alpha} + 2\zeta_\xi \frac{\tilde{\omega}}{U^*}\dot{\xi} + \left(\frac{\tilde{\omega}}{U^*}\right)^2 G(\xi) = -\frac{1}{\pi\mu} C_L(\tau), \\ &\frac{x_\alpha}{r_\alpha^2}\ddot{\xi} + \ddot{\alpha} + 2\frac{\zeta_\alpha}{U^*}\dot{\alpha} + \frac{1}{(U^*)^2} M(\alpha) = \frac{2}{\pi\mu r_\alpha^2} C_M(\tau), \end{aligned} \tag{4.1}$$

where ξ denotes the plunging displacement (positive downwards), α is the pitch angle about the elastic axis (positive nose up), $G(\xi)$ and $M(\alpha)$ are the nonlinear plunge and pitch stiffness terms, respectively, and $C_L(\tau)$, $C_M(\tau)$ are the lift and moment coefficients due to the aerodynamic forces. For a subsonic flow, $C_L(\tau)$ and $C_M(\tau)$ can be expressed by the integral formulas in [140]. The structural nonlinearities are represented by $G(\xi)$ and $M(\alpha)$. In the case of a *cubic spring*:

$$M(\alpha) = \beta_0 + \beta_1\alpha + \beta_2\alpha^2 + \beta_3\alpha^3, \tag{4.2}$$

where β_0, β_1, β_2, β_3 are constants. For a *freeplay model*:

$$M(\alpha) = \begin{cases} M_0 + \alpha - \alpha_f & \text{if } \alpha < \alpha_f \\ M_0 + M_f(\alpha - \alpha_f) & \text{if } \alpha_f \leq \alpha \leq \alpha_f + \delta \\ M_0 + \alpha - \alpha_f + \delta(M_f - 1) & \text{if } \alpha > \alpha_f + \delta \end{cases} \tag{4.3}$$

where M_0, M_f, α_f and δ are constants. Here we give the expression for $M(\alpha)$ in the pitch degree of freedom. Similar expressions for $G(\xi)$ in the plunge motion can be written by replacing α with ξ [140, 151].

The solution of the above integro-differential equations can be obtained by solving a reformulated system of ordinary differential equations in $\mathbf{R}^8$ [140]. The system parameters are chosen such that the resulting aeroelastic response — numerically generated by using a fourth-order Runge-Kutta time integration scheme [151] with respect to the non-dimensional time τ (defined as in [140]) — corresponds to a limit cycle oscillation. The signals S1, S3, S5 (see Figs.4.1-4.3) represent the time history of the pitch motion for the aeroelastic system, scaled to amplitude ≈1, for different values of the system parameters. S1 was generated by using cubic stiffness terms in both the pitch and plunge degrees of freedom, while in S3 and S5 the structural nonlinearity is represented by a freeplay model in the pitch degree-of-freedom and a linear spring in the plunge degree-of-freedom [238, 239]. Signals S2, S4, S6 are obtained by contaminating S1, S3, S5 respectively with additive noise normally distributed with mean zero and signal-to-noise ratio equal to 5.

In Figs.4.4-4.6, the time histories of both the clean and noisy training set are represented for each test case. The left-hand and right-hand limits are $t_0 - n_0 + 1$ and t_1 respectively, while the

	S1	S2	S3	S4	S5	S6	S7	S8
n_0	100	100	50	50	90	90	20	20
n_1	2	2	2	3	2	3	2	3
$t_0 - n_0$	50	50	0	0	0	0	175	175
t_1	273	325	282	325	380	380	300	305

Table 4.1: ANN Architecture and Training Parameters.

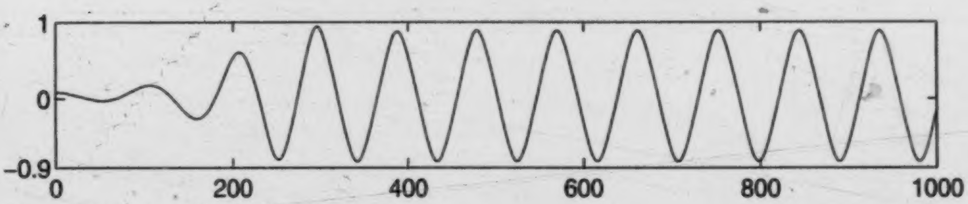

Figure 4.1: Test Signal S1.

vertical bar is at $t_0 + 1$. Note that the number n_0 of ANN inputs is always chosen to be roughly equal to the length of an oscillation cycle in the training set. The values of t_0, t_1, and n_0 are also recorded in Table 4.1. The phase plots of the clean training sets ($s(t)$ vs. $s(t-d)$, $t_0+1 \le t \le t_1$, for selected lags d) are also illustrated in Figs.4.4-4.6. The small circle corresponds to the first data point $[s(t_0 - d + 1), s(t_0 + 1)] \in \mathbf{R}^2$.

Unlike the simulated data sets S1-S6, the signals S7-S8 represent experimental data recorded in a wind tunnel experiment reported by Ko, Strganac, Kurdila, et. al. The data is available at http://aerounix.tamu.edu/aeroel/ and the details of the experiment are described in [21, 124, 125, 132, 133, 222]. In these experiments, a wing with a control surface is mounted to allow plunge (h) and pitch (α) motions about the elastic axis. The motion of the system can be described by the following model:

$$\begin{bmatrix} m_T & m_W x_\alpha b \\ m_W x_\alpha b & I_\alpha \end{bmatrix} \begin{bmatrix} \ddot{h} \\ \ddot{\alpha} \end{bmatrix} + \begin{bmatrix} c_h & 0 \\ 0 & c_\alpha \end{bmatrix} \begin{bmatrix} \dot{h} \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} k_h & 0 \\ 0 & k_\alpha \end{bmatrix} \begin{bmatrix} h \\ \alpha \end{bmatrix} = \begin{bmatrix} -L \\ M \end{bmatrix} \tag{4.4}$$

In the above system, m_W is the mass of the wing, m_T is the total mass of the wing and its support structure, x_α is the nondimensional distance between the center of mass and the elastic

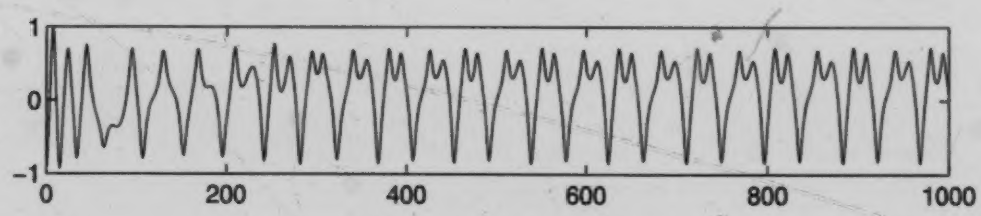

Figure 4.2: Test Signal S3.

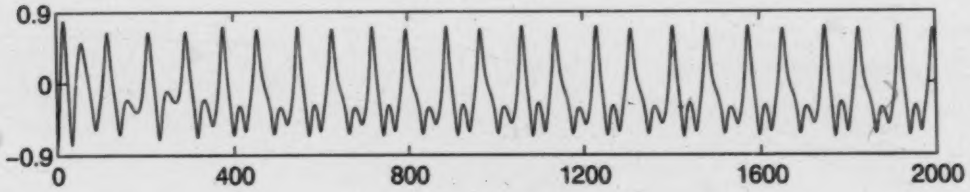

Figure 4.3: Test Signal S5.

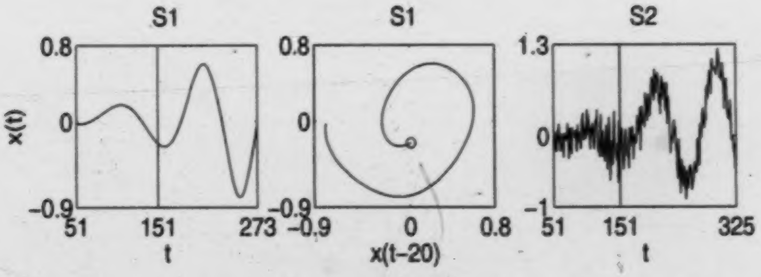

Figure 4.4: Training Sets: Time History and Phase Plot for S1, Time History for S2.

axis of the wing, I_α is the mass moment of inertia about the elastic axis, c_h, c_α are the plunge and pitch structural damping coefficients respectively, k_h, k_α are the structural stiffnesses for the plunge and pitch motion respectively, L and M are the aerodynamic lift and moment about the elastic axis. The nonlinearity in the sytem is provided by the torsional stiffness, which is approximated in polynomial form as: $k_\alpha(\alpha) = k_{\alpha_0} + k_{\alpha_1}\alpha + k_{\alpha_2}\alpha^2 + k_{\alpha_3}\alpha^3 + k_{\alpha_4}\alpha^4 + \dots$ [21, 124, 125, 132, 133, 222].

The signal S7 (see Fig.4.7) represents the time history of the pitch motion and was obtained by selecting every 10th point of the pitch signal in the file DN04J.DAT and scaling the resulting data set to amplitude ≈ 1 [238, 239]. Signal S8 is obtained by contaminating S7 with additive noise normally distributed with mean zero and signal-to-noise ratio equal to 5. In Fig.4.8, the time histories of both the clean and noisy training set are illustrated. The left-hand and right-hand limits indicate $t_0 - n_0 + 1$ and t_1 respectively, and the vertical bar is at $t_0 + 1$. The values

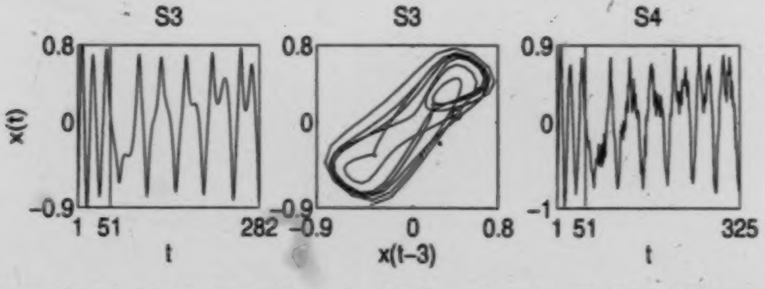

Figure 4.5: Training Sets: Time History and Phase Plot for S3, Time History for S4.

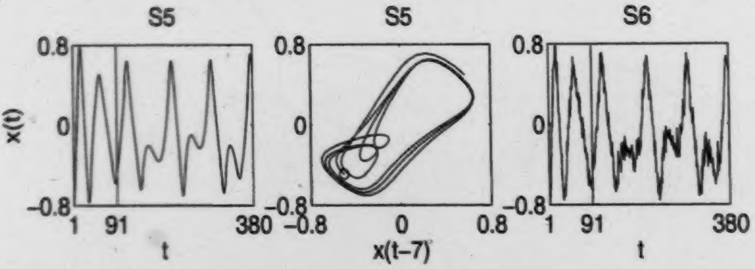

Figure 4.6: Training Sets: Time History and Phase Plot for S5, Time History for S6.

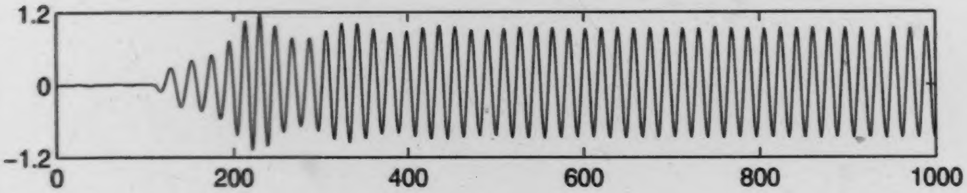

Figure 4.7: Test Signal S7.

of t_0, t_1, and n_0 are also recorded in Table 4.1. The phase plot of the clean training set is also illustrated in Fig.4.8.

The PPs of the training sets show that, for the majority of the test signals, the successive oscillation cycles are close together. Hence, it is expected that the trajectory converges to a limit cycle in each case. Therefore, Δ in Propositions 3.4.20 and 3.4.24 is expected to be small, and it follows that $\tilde{b}^{ts}_{\mathbf{w}}$ will be small after ANN training, i.e., the OSP throughout the given trajectory will be accurate (see Remark 3.4.3). Then, by Remark 3.4.5, it is expected that $\tilde{\epsilon}_{\mathbf{w}}$ in Definition 3.4.22 will be small. In other words, by Proposition 3.4.34, it is expected that the ω-limit set of the MSP will be close to the ω-limit set of the given trajectory. The only exception are the signals S1-2, for which we cannot draw any conclusion from the inspection of the training set. The given trajectory might very well exhibit a divergent oscillation instead of a limit cycle in those cases.

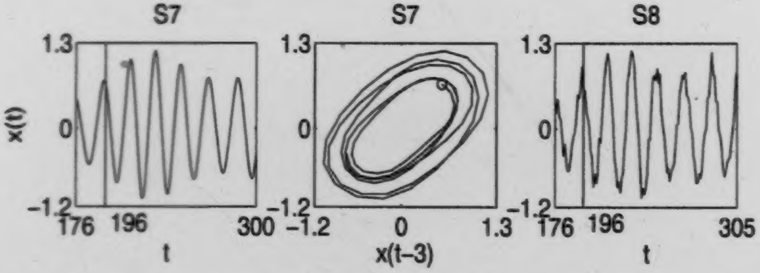

Figure 4.8: Training Sets: Time History and Phase Plot for S7, Time History for S8.

Feature of Interest	Digit	0	1	2	3
2nd Layer Transf. Fct.	1st	N/A	Linear	Linear	Scal.Tanh.
Second Layer Weights	1st	N/A	Unscaled	Scaled	Unscaled
Init. $w_0^{(2)}, w_{k_1}^{(2)}, w_{k_1,0}^{(1)}$	2nd	All Zero	All Zero	N/A	N/A
Init. $w_{k_1,k_0}^{(1)}, k_0 \neq 0$	2nd	Sig.Segm.	Random	N/A	N/A
Learning Rate Type	3rd	Adaptive	Constant	N/A	N/A

Table 4.2: ANN MSP Methods

4.2 Choice of Methods and Parameters

Twelve LTMSP methods are presented, and each LTMSP method will be identified by a code number consisting of three digits. The significance of each digit will be explained in the following.

The first digit in the method code is '1' if a classical 2LFF1S2LNN with $f^{(1)}(x) = \tanh(x)$ is used. The ANN output in that case is given by (3.26) and (3.27) (see Fig.3.1). The first digit is set to '2' if a 2LFF1S2LNN with scaled weights in the second layer is used. The ANN output in that case will be given by (3.31). The first digit in the method's code is '3' if a 2LFF1S2SNN with $f^{(2)}(x) = \psi(w^{(3)})\tanh(x)$ is used. The ANN output in that case will be given by (3.29).

The second digit is '1' if all $w_{k_1,k_0}^{(1)}, k_1 \neq 0, k_0 \neq 0$ are initialized with random numbers uniformly distributed in $[-0.01, +0.01]$, while all the other weights are initially set to zero. The second digit is '0' if $w_{k_1,k_0}^{(1)}, k_1 \neq 0, k_0 \neq 0$ are initially set to normalized segments of the training set, as in (3.47), while all the other weights are initialized by zero.

The third digit is '1' if the learning rate (LR) is kept constant during ANN training, and '0' if the LR is adaptive, i.e., is updated at each training iteration. The appropriate values for the constant LR (CLR) or the initial value of the adaptive upper bound on the adaptive LR (ALR) are chosen as described in Section 3.3.

A summary of the methods used is presented in Table 4.2, where for each feature of the ANN MSP method, the digit in the method's code relevant to that feature is indicated, as well as the significance of each value that the respective digit may take. For example, M110 refers to a 2LFF1SNN with linear TF in the second layer, in which no scaling is employed for the second-layer weights. Moreover, random numbers are used to initialize the first-layer weights and an adaptive learning rate is adopted in training. It should be noted that M110 is actually one of the most commonly used networks in the ANN literature.

For each case study, the number of ANN inputs n_0 needs to be chosen. To do that, we will follow the procedure described in Section 3.3. First, an empirical estimate n_0^{emp} is computed based on the average distance between consecutive Lmax (or Lmin) points in the training set. In each case, the vector of consecutive Lmax points $\mathbf{t}^{\text{Lmax}}$, the vector of the distances between consecutive Lmax points $\mathbf{d}^{\text{Lmax}}$, and the average distance between consecutive Lmax points

Sig	Type	Components of **t**	Components of **d**	a
S1	Lmin	53, 159, 252	106, 93	100
S1	Lmax	106, 207	101	101
S2	Lmin	50, 140, 250	90, 110	100
S2	Lmax	100, 205, 295	105, 90	98
S3	Lmin	108, 153, 196, 242	45, 43, 46	45
S4	Lmin	108, 153, 197, 242, 283	45, 44, 45, 41	44
S5	Lmax	114, 208, 292, 376	94, 84, 84	87
S6	Lmax	113, 208, 291, 375	95, 83, 84	87
S7	Lmin	204, 221, 238, 257, 277, 297	17, 17, 19, 20, 20	19
S7	Lmax	195, 213, 230, 247, 267, 287	18, 17, 17, 20, 20	18
S8	Lmin	203, 222, 237, 257, 276, 296	19, 15, 20, 19, 20	19
S8	Lmax	195, 213, 230, 248, 268, 287	18, 17, 18, 20, 19	18

Table 4.3: Local Minimum and Local Maximum Points in the Test Signals.

a^{Lmax}, can be defined. The same can be done for the Lmin points. For each test signal, the type of local extremum points considered (Lmin or Lmax), the components of $\mathbf{t}^{\mathrm{Lmin}}$ and/or $\mathbf{t}^{\mathrm{Lmax}}$, the components of $\mathbf{d}^{\mathrm{Lmin}}$ and/or $\mathbf{d}^{\mathrm{Lmax}}$, and the averages a^{Lmin} and/or a^{Lmax}, are reported in Table 4.3. The values of the estimate n_0^{emp} of n_0 obtained in each case are reported in Table 4.4. Note that n_0^{emp} is the average of a^{Lmin} and a^{Lmax} if both Lmax and Lmin points have been considered, and $n_0^{\mathrm{emp}} = a^{\mathrm{Lmin}}$ (or $n_0^{\mathrm{emp}} = a^{\mathrm{Lmax}}$, respectively) if only Lmin points (or Lmax points, respectively) have been considered. A more detailed explanation will be provided in the following.

For S1 and S2, the entire known data set $s(1), s(2), \ldots, s(t_1)$, was used to calculate n_0^{emp}, and both the Lmin and Lmax points in the data set were considered. For these test cases, $n_0^{\mathrm{emp}} = \left(a^{\mathrm{Lmin}} + a^{\mathrm{Lmax}}\right)/2$. For S3 and S4, due to the more complicated profile of the training set, which contains a high-frequency component, only the last 4 (and 5, respectively) consecutive Lmin points in the training set have been used to compute n_0^{emp}, which in these cases is equal to a^{Lmin}. For the same reason, in the case of S5 and S6, only the last 4 consecutive Lmax points in the training set have been used to compute n_0^{emp}, which in these cases is equal to a^{Lmax}. The test signals S7 and S8 (see Fig.4.7) initially exhibit an oscillation of lower frequency than the one of the segment used for ANN training. Actually, the oscillation frequency increases with the time starting from $t = 111$ up to approximately $t = 190$. Therefore only the data points $s(190), s(191), \ldots, s(t_1)$ (where $t_1 = 300$ and 305, respectively) were used to compute n_0^{emp}. Both the Lmin and Lmax points in the data set were considered, therefore again $n_0^{\mathrm{emp}} = \left(a^{\mathrm{Lmin}} + a^{\mathrm{Lmax}}\right)/2$ for S7 and S8.

Next, the value of n_0^{acf} for each test case needs to be computed. The estimate of the ACF in

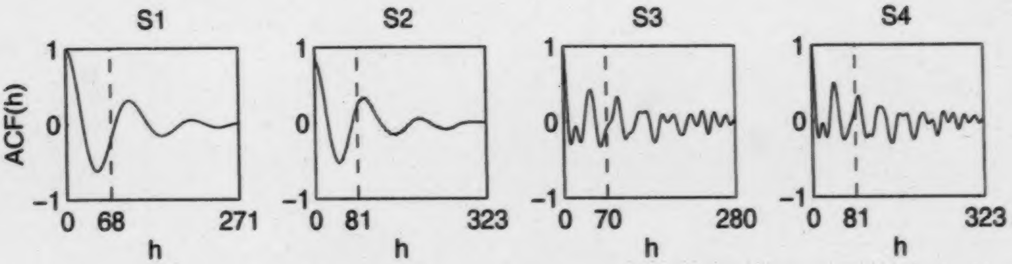

Figure 4.9: ACF plot for S1-S4.

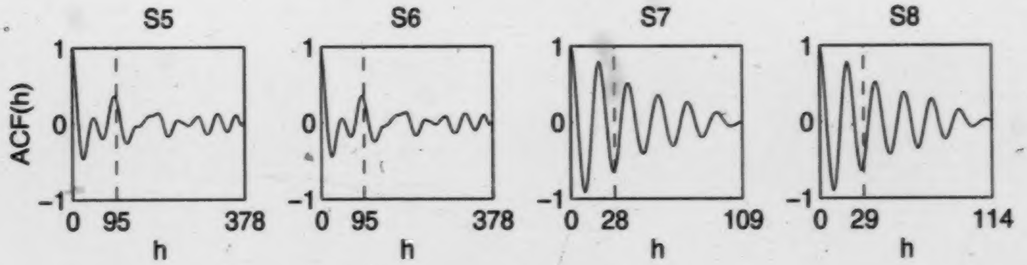

Figure 4.10: ACF plot for S5-S8.

each case was computed as in (3.49), and the functions are plotted in Figs.4.9-4.10. The dashed vertical bar in all graphs marks the lag $h = (t_1 - t_0)/4$, up to which the ACF estimates are expected to be accurate. The values of n_0^{acf} (i.e., the value of h for which the first peak of $\hat{\gamma}(h)$ is achieved) for each test case are reported in Table 4.4. For S1-S6, the entire known data set $s(1)$, $s(2)$, …, $s(t_1)$, was used to calculate n_0^{acf}, while for S7 and S8 only the data points $s(190)$, $s(191)$, …, $s(t_1)$ were used, for the same reasons as mentioned above.

Finally, the estimate n^{fft} was calculated. In each case, the FFT of the training set was computed, assuming the sampling step to be 1. The graphs are shown in Figs.4.11-4.14. The dashed vertical bar marks the lowest frequency ν_1 that has an energy close to the maximum energy of all Fourier modes in the decomposition. The values of $n_0^{\mathrm{fft}} = 1/\nu_1$ for each test case are reported in Table 4.4. For S1-S6, the entire known data set $s(1)$, $s(2)$, …, $s(t_1)$, was used to calculate n_0^{fft}, while for S7 and S8 again only the data points $s(190)$, $s(191)$, …, $s(t_1)$ were used.

The values of $n_0^{\max} = \max\left\{n_0^{\mathrm{emp}}, n_0^{\mathrm{acf}}, n_0^{\mathrm{fft}}\right\}$ and $n_0^{\mathrm{avg}} = \left(n_0^{\mathrm{emp}} + n_0^{\mathrm{acf}} + n_0^{\mathrm{fft}}\right)/3$, as well as our final choice n_0^{fin} of the number of ANN inputs, are also reported in Table 4.4. The value of n_0^{fin} was obtained, in general, by rounding off $n_0^{\max}$ and n_0^{avg} (which had similar values) to the closest multiple of 10. For S3 and S4, due to the more complicated profile of the signals, we rounded off the estimates to 50 ANN inputs instead of 40. The values of the final number of ANN inputs chosen in each case ($n_0 = n_0^{\mathrm{fin}}$) are also reported in Table 4.1.

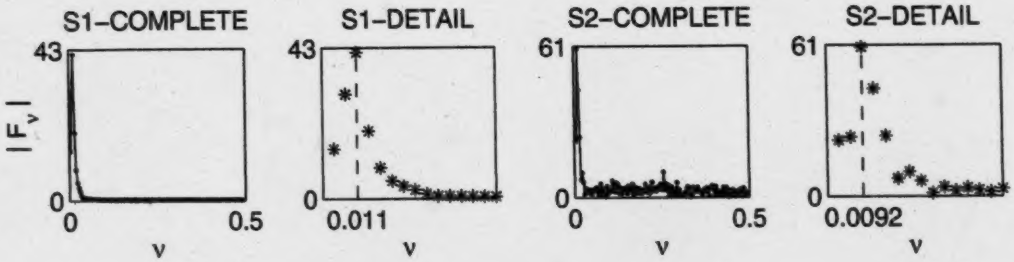

Figure 4.11: FFT plot for S1 and S2.

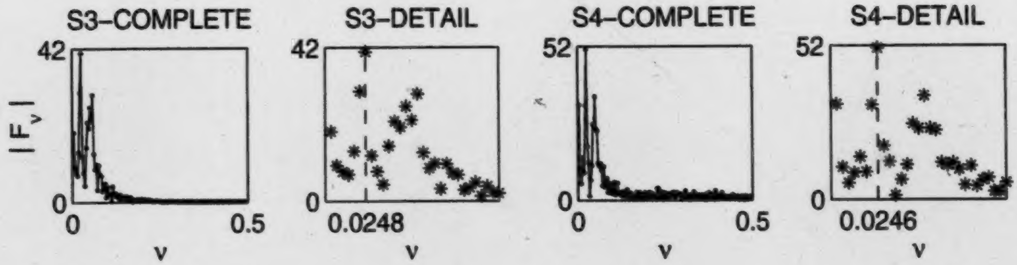

Figure 4.12: FFT plot for S3 and S4.

For each test case, the selection of the number of hidden neurons n_1 is described in the following. Starting from $n_1 = 2$, increment n_1 until at least one of the 12 methods provides a good MSP of the given signal using n_1 neurons in the hidden layer. In general, we observed that 2 hidden neurons were sufficient for the clean signals while 3 hidden neurons were necessary for predicting the noisy signals. The only exception is the noisy signal S2, for which 2 hidden neurons were sufficient, due to the existence of a clear low-frequency oscillation that makes it easy for the ANN to separate the noise from the underlying clean signal. Once n_1 is fixed for the current test case, each of the 12 methods is applied to the signal using n_1 neurons in the hidden layer. The values of n_1 are recorded in Table 4.1.

4.3 Prediction Accuracy

The THs and the phase plots for selected lags d based on the ANN-generated MSPs for the 96 test cases are shown in Figs.4.15-4.58 at the end of this chapter. The prediction performance of the 12 ANN MSP methods is compared in the following manner. For each test case, the MSP result generated by the ANN after the end of training is investigated. The prediction accuracies are compared based on the TH and PP overlapping, respectively. A prediction accuracy score (PAS) between 0 and 10 is assigned to each test case. The test cases that provide the best MSP

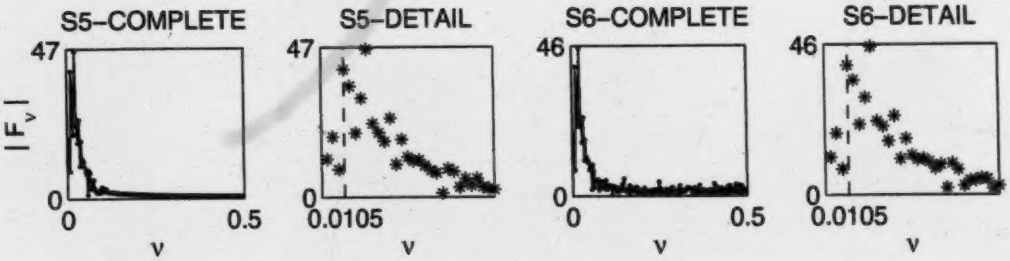

Figure 4.13: FFT plot for S5 and S6.

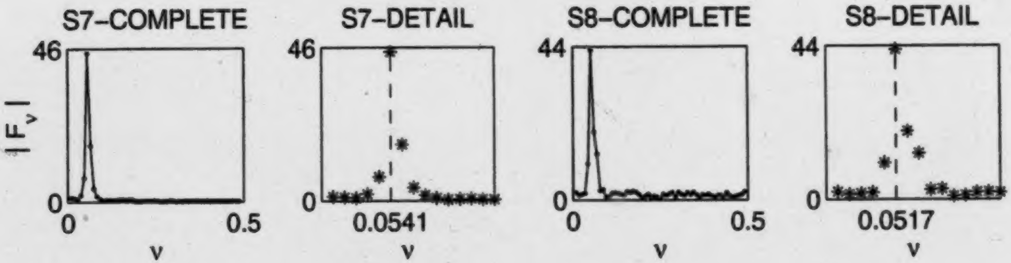

Figure 4.14: FFT plot for S7 and S8.

based on TH overlapping, of all 96 test cases, will receive a TH-based PAS (THPAS) of 10, the next best ones a score of 9, etc. Similarly, the test cases that provide the best MSP based on PP overlapping receive a PP-based PAS (PPPAS) of 10, the next best ones a score of 9, etc. The test cases for which the ANN training did not converge at all (hence no MSP was provided) received a PAS of 0. The THPAS and PPPAS were averaged for each test case, generating an overall PAS (OAPAS) in each case. The THPAS, PPPAS, and OAPAS values are recorded in Tables 4.5, 4.6, and 4.7 respectively. In the columns labeled 'Clean' and 'Noisy', the average PASs over all clean (S1, S3, S5, S7) and noisy (S2, S4, S6, S8) signals respectively, are reported. In the column 'Diff', the differences between these averages are shown, providing a measure of the sensitivity to noise of each ANN MSP method. The column 'All' reports the overall average PASs over all 8 signals for each of the 12 methods. It was noticed in our experiments that, even though for a particular test signal the THs may be perfectly matched while the PPs are not, or viceversa, both prediction accuracy criteria reveal the same average behavior of the different ANN MSP methods.

From Table 4.7, it is obvious that for the MXX0 methods, involving an ALR, the ANN training is almost always unstable when performed on S4-S6 and S8. In two cases, even if the training is stable, the MSP accuracy is very poor (OAPAS=2) for these methods. As it can be seen from the 'Diff' column, the MXX0 methods are very sensitive to noise in the training signal,

	S1	S2	S3	S4	S5	S6	S7	S8
t_I	1	1	1	1	1	1	190	190
t_F	273	325	282	325	380	380	300	305
n_0^{emp}	101	99	45	44	87	87	19	19
n_0^{acf}	98	94	43	42	90	91	18	18
n_0^{fft}	91	109	40	41	95	95	18	19
n_0^{max}	101	109	45	44	95	95	19	19
n_0^{avg}	97	101	43	42	91	91	18	19
n_0^{fin}	100	100	50	50	90	90	20	20

Table 4.4: Selection of Network Inputs

which is not surprising since an ALR search algorithm is more versatile than one using a CLR, and therefore makes it possible for the ANN to learn the noise immediately, thus destroying the underlying pattern. Even in the case of the clean freeplay signal S5, the ANN training failed to converge for most MXX0 methods. S5 is particularly difficult to predict due to the very short transient data set and to its complicated profile. In addition, S3 and S5 (and thus S4, S6 as well) contain numerical errors due to the fact that they have been generated by using a fourth-order Runge-Kutta time-integration scheme in a system with a nonsmooth nonlinearity [151].

Obviously, the MX01 methods exhibit a better behaviour than the other methods. In particular, for the test signal S8, they are the only ones for which the ANN training converges. Signal S8 is considered to be the most difficult to predict, because it represents real-life measurement data contaminated by measurement noise as well as by mathematically generated additive noise. The MX01 methods provide excellent predictions in this case. Overall, considering all 12 test cases, MX01 also provide more accurate predictions. M201 and M101 have practically the same performance (OAPAS$\approx$8.5) and they are both robust (in addition, M201 is less sensitive to noise than M101). M301 provides an OAPAS$\approx$7 but its behaviour is less consistent throughout the 12 test cases.

It is interesting to note that M301 is the only method out of the 12 considered for which the ANN training is unstable in the case of a clean signal (S5) while it becomes stable when noise is introduced (S6). For all the other methods, the training either converges for both the clean and the noisy signal, or it does not converge for both signals, or it converges for the clean signal but not for the noisy signal. It is well known that, for the interpolative applications, adding noise in the ANN input and/or output training prototypes actually improves the generalization performance of the network [247]. Multi-step prediction, however, is more than an interpolation problem. In order to provide an accurate LTMSP, a good interpolation as well as little sensitivity of the ANN output with respect to variations in the inputs is required. It seems that, due to the additional sigmoidal transfer function, that limits the damage caused by noise, the ANN is able

	S1	S2	S3	S4	S5	S6	S7	S8	Clean	Noisy	Diff	All
M100	7	8	9	0	0	0	8	0	6.00	2.00	4.00	4.00
M101	10	7	10	7	10	10	10	10	10.00	8.50	1.50	9.25
M110	7	7	9	0	3	0	8	0	6.75	1.75	5.00	4.25
M111	7	8	10	5	6	6	9	0	8.00	4.75	3.25	6.38
M200	7	8	9	0	0	0	8	0	6.00	2.00	4.00	4.00
M201	10	8	10	10	10	10	10	10	10.00	9.50	0.50	9.75
M210	6	7	9	0	3	0	8	0	6.50	1.75	4.75	4.13
M211	7	8	8	5	3	10	9	0	6.75	5.75	1.00	6.25
M300	7	8	10	0	0	0	8	0	6.25	2.00	4.25	4.13
M301	10	8	8	7	0	6	8	10	6.50	7.75	-1.25	7.13
M310	7	7	10	0	0	0	9	0	6.50	1.75	4.75	4.13
M311	7	7	10	5	4	6	10	0	7.75	4.50	3.25	6.13

Table 4.5: Time History -based Prediction Accuracy Scores (THPAS).

to improve its generalization capabilities. In the cases of S5 and S6, the additive noise actually helps the ANN training to converge to a not-so-bad solution, by compensating for the numerical errors already present in the signal.

MX11 provide practically the same (OAPAS$\approx$5.75) prediction accuracy. M111 and M311 are less sensitive to noise as MXX0, but much more sensitive than MX01. M211 is as robust in the presence of noise as M201. As one would expect, normalizing the weights in the second layer provides robustness with respect to the noise and to variations in the ANN parameters.

4.4 Robustness Comparison

In the previous section, it has been shown that the MSP methods M101 and M201 provide the best prediction accuracy. In this section, we compare the stability of the two methods with respect to the variations in the number of inputs n_0, the hidden neurons n_1, and the endpoint t_1 of the training set. First of all, by varying n_0 or n_1, the number of parameters (weights) in the neural network varies. This will affect the distribution of information in the network and will result in different ANN models. When t_1 changes, the size of the training set (hence the amount of available information) changes, as well as the starting point of the MSP process. It is known that for a nonlinear time series the expected OSP accuracy (and hence the expected MSP accuracy) depends on time, unlike the case of linear time series [68]. Due to the nonlinear nature of the considered signals, the dependence of the ANN-generated MSP on n_0 and t_1 is expected to be nonlinear.

For each of the two methods applied to the 8 test signals, each of the three parameters of

	S1	S2	S3	S4	S5	S6	S7	S8	Clean	Noisy	Diff	All
M100	7	6	6	0	0	0	10	0	5.75	1.50	4.25	3.63
M101	9	6	6	6	7	7	10	10	8.00	7.25	0.75	7.63
M110	9	6	6	0	1	0	10	0	6.50	1.50	5.00	4.00
M111	7	6	6	5	3	7	10	0	6.50	4.50	2.00	5.50
M200	8	6	6	0	0	0	10	0	6.00	1.50	4.50	3.75
M201	9	6	6	6	7	7	10	10	8.00	7.25	0.75	7.63
M210	3	6	6	0	1	0	10	0	5.00	1.50	3.50	3.25
M211	4	6	6	5	1	7	10	0	5.25	4.50	0.75	4.88
M300	4	5	6	0	0	0	10	0	5.00	1.25	3.75	3.13
M301	9	5	6	6	0	7	10	10	6.25	7.00	-0.75	6.63
M310	4	5	6	0	0	0	10	0	5.00	1.25	3.75	3.13
M311	4	5	6	5	3	7	10	0	5.75	4.25	1.50	5.00

Table 4.6: Phase Portrait -based Prediction Accuracy Scores (PPPAS).

interest was varied while keeping the other two constant. The degree of variability of the MSPs caused by the variation of each parameter was assessed by inspecting both the THs and the PPs of the predicted signals. The goal is to determine which of the two methods M101 and M201 more robustly extracts information from the training set using the minimum number of hidden neurons. It must be noted that this is a borderline situation, since the minimum number of neurons is used in the ANN. The more hidden neurons there are in the ANN, the more robust it becomes with respect to variations in n_0, n_1, and t_1. It has been noticed that even when using hundreds of hidden neurons, overfitting does not occur when this type of weight initialization is used in combination with a constant learning rate in ANN training.

For each test case, the THs of the last two oscillation cycles of the ANN-generated MSPs, as well as the PPs for selected lags, for both M101 and M201, are displayed in Figs.4.59-4.85. Let n_0^*, n_1^*, t_1^* be the values of n_0, n_1, t_1 respectively used in the first part of this study, in which the prediction accuracies of the 12 methods were compared. In the robustness analysis, we vary the three parameters as follows. For $n_1 = n_1^*$ and $t_1 = t_1^*$: $n_0 = 0.8 \times n_0^*$, $0.9 \times n_0^*$, n_0^*, $1.1 \times n_0^*$, $1.2 \times n_0^*$ (see Figs.4.59-4.67). For $n_0 = n_0^*$ and $t_1 = t_1^*$: $n_1 = n_1^*$, $n_1^* + 1$, $n_1^* + 2$, $n_1^* + 3$ (see Figs.4.68-4.75). For $n_0 = n_0^*$ and $n_1 = n_1^*$: $t_1 = t_1^* - 0.2 \times n_0^*$, $t_1^* - 0.1 \times n_0^*$, t_1^*, $t_1^* + 0.1 \times n_0^*$, $t_1^* + 0.2 \times n_0^*$ (see Figs.4.76-4.85). In most experiments it was noticed that, for each test signal, as one of the 3 parameters is varied, the MSP results corresponding to different values of that parameter are more or less shifted in time with respect to each other, but they all exhibit a limit cycle with the same profile. The PPs of the predicted signals do not vary significantly as the parameter changes. In most cases, M101 and M201 behave identically when a parameter is varied.

	S1	S2	S3	S4	S5	S6	S7	S8	Cln.	Nsy.	Dif.	All
M100	7.0	7.0	7.5	0.0	0.0	0.0	9.0	0.0	5.88	1.75	4.13	3.81
M101	9.5	6.5	8.0	6.5	8.5	8.5	10.0	10.0	9.00	7.88	1.13	8.44
M110	8.0	6.5	7.5	0.0	2.0	0.0	9.0	0.0	6.63	1.63	5.00	4.13
M111	7.0	7.0	8.0	5.0	4.5	6.5	9.5	0.0	7.25	4.63	2.63	5.94
M200	7.5	7.0	7.5	0.0	0.0	0.0	9.0	0.0	6.00	1.75	4.25	3.88
M201	9.5	7.0	8.0	8.0	8.5	8.5	10.0	10.0	9.00	8.38	0.63	8.69
M210	4.5	6.5	7.5	0.0	2.0	0.0	9.0	0.0	5.75	1.63	4.13	3.69
M211	5.5	7.0	7.0	5.0	2.0	8.5	9.5	0.0	6.00	5.13	0.88	5.56
M300	5.5	6.5	8.0	0.0	0.0	0.0	9.0	0.0	5.63	1.63	4.00	3.63
M301	9.5	6.5	7.0	6.5	0.0	6.5	9.0	10.0	6.38	7.38	-1.00	6.88
M310	5.5	6.0	8.0	0.0	0.0	0.0	9.5	0.0	5.75	1.50	4.25	3.63
M311	5.5	6.0	8.0	5.0	3.5	6.5	10.0	0.0	6.75	4.38	2.38	5.56

Table 4.7: Overall Prediction Accuracy Scores (OAPAS).

There are exceptions from this stable behaviour, which will be discussed below. In some cases, M201 exhibits much better stability than M101 when a parameter of interest is varied. This phenomenon occurs for S8 when n_0 is varied (see Fig.4.67), for S4 when n_1 is varied (see Fig.4.71), and for S3 (see Fig.4.79) and S6 (see Fig.4.83) when t_1 is varied. In four cases (S2: $n_0 = 1.1 \times n_0^*$: see Fig.4.60, S4: $n_0 = 1.1 \times n_0^*$ and $n_0 = 1.2 \times n_0^*$: see Fig.4.62, S2: $t_1 = t_1^* - 0.1 \times n_0^*$: see Fig.4.78), M201 provides a stable prediction while the ANN training for M101 does not converge at all. In two cases (S4: $t_1 = t_1^* + 0.1 \times n_0^*$, S5: $n_0 = 0.9 \times n_0^*$), the ANN training did not converge for either of M101, M201. For S6 when $n_0 = 0.8 \times n_0^*$ (see Fig.4.65), for S2 when $t_1 = t_1^* - 0.2 \times n_0^*$ (see Fig.4.77), and for S5 when $t_1 = t_1^* + 0.2 \times n_0^*$ (see Fig.4.82), both M101 and M201 provided equally inaccurate predictions (in the last case, M201 actually performed a little worse than M101). The only case when M201 clearly performed worse than M101 was for S5: $n_0 = 0.8 \times n_0^*$ (see Fig.4.63), when the network training for M201 did not converge, while M101 did provide an acceptable prediction.

From the above results, it is obvious that the MSP depends nonlinearly on n_0 and t_1, which is to be expected since the test signals exhibit nonlinear dynamics. The stability of both methods with respect to t_1 is slightly better than with respect to n_0. When only n_0 is varied, the nonlinear mapping $\tilde{\Phi}$ that models the dependence of $s(t)$ on the past n_0 observations throughout the training set is expected to change dramatically. When only t_1 is varied, the mapping $\tilde{\Phi}$ to be approximated by $\Phi_{\mathbf{w}}$ remains the same, while only the amount of training data is changed. Since t_1 is altered by no more than 20% of an oscillation cycle, the mapping $\Phi_{\mathbf{w}}$ obtained by least-square optimization cannot vary too much. Thus, the contribution of 20% of an oscillation cycle is small when errors are averaged over the entire training set. What may vary significantly with t_1 is the so-called 'predictability' — the conditional variance of the OSP error $\tilde{e}_{\mathbf{w}}(t_1+1)$ [68] — at the moment t_1, variation caused by the nonlinear nature of the time series. In the case of a nonlinear time series, the above quantity is difficult, if not impossible, to estimate. A possible solution would be to consider all MSP signals starting in every moment throughout the last known oscillation cycle: $t_1 - n_0 + 1, \ldots, t_1$, rather than only the MSP signal starting at the monment t_1 (as in (3.21), (3.22)):

$$\hat{\mathbf{y}}_{\mathbf{w}}^{<h>}(t) \overset{\text{def}}{=} \Phi_{\mathbf{w}}\left(\hat{\mathbf{y}}_{\mathbf{w}}^{<h>}(t-1)\right), \ \forall t \geq t_1 - h + 2, \ \hat{\mathbf{y}}_{\mathbf{w}}^{<h>}(t_1 - h + 1) \overset{\text{def}}{=} \mathbf{s}(t_1 - h + 1) \tag{4.5}$$

for $1 \leq h \leq n_0$. Overall, it is clear enough that, under neuron scarcity constraints, M201 is much more robust than M101 with respect to variations in the number of network inputs and hidden neurons, as well as with respect to variations in the endpoint of the training set.

It should be noticed in practice that an ANN with more hidden neurons exhibits better stability with respect to variations in n_1, n_0, and t_1 than an ANN with fewer hidden neurons. More specifically, suppose t_1 and n_0 are fixed. If the ANN training converges for some n_1^I and also for some $n_1^{II} \gg n_1^I$, then the second ANN will be more stable with respect to variations in n_1, n_0, and t_1 than the first ANN. This phenomenon is due to the redundant information storage in an ANN. The more redundancy there is in the network, the more stable it is with respect to variations in the different parameters of the method. Our experiments demonstrate that, even though the performance of some of the the discussed methods may improve if more hidden neurons are used [259], the method that robustly extracts the most information from the known data set using the smallest number of hidden neurons is the one using second-layer weight scaling, signal-related weight initialization and a constant learning rate in network training (M201).

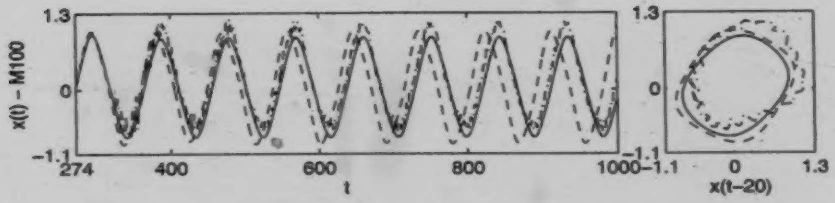

Figure 4.15: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M100.

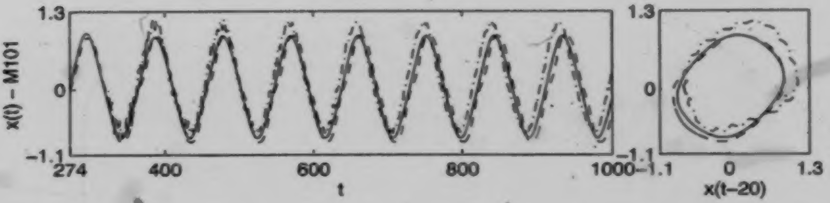

Figure 4.16: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M101.

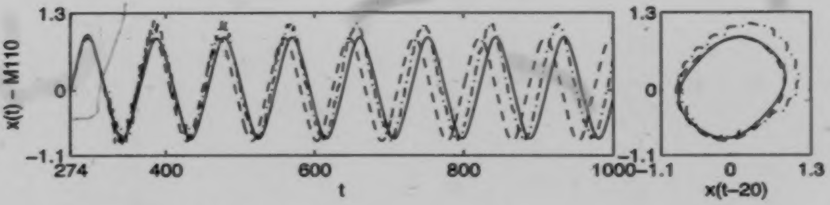

Figure 4.17: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M110.

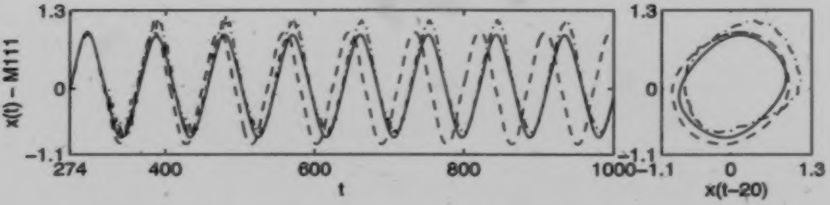

Figure 4.18: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M111.

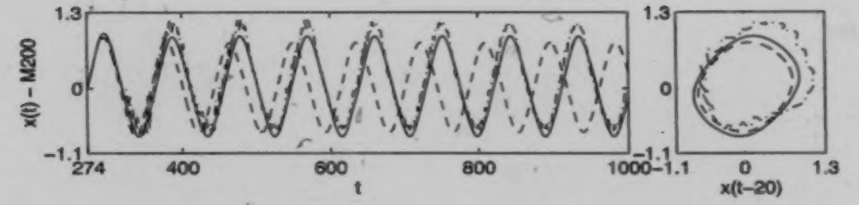

Figure 4.19: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M200.

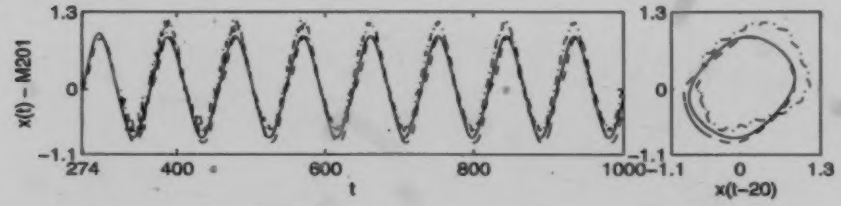

Figure 4.20: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M201.

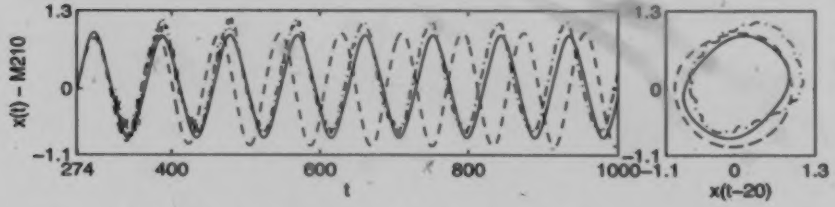

Figure 4.21: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M210.

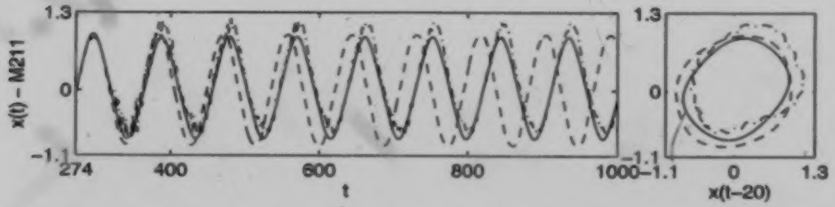

Figure 4.22: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M211.

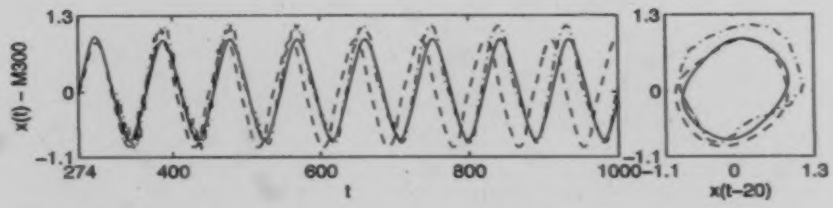

Figure 4.23: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M300.

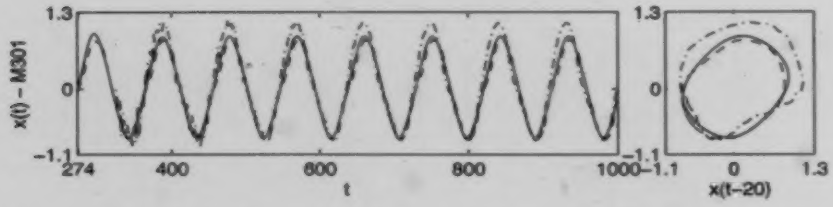

Figure 4.24: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M301.

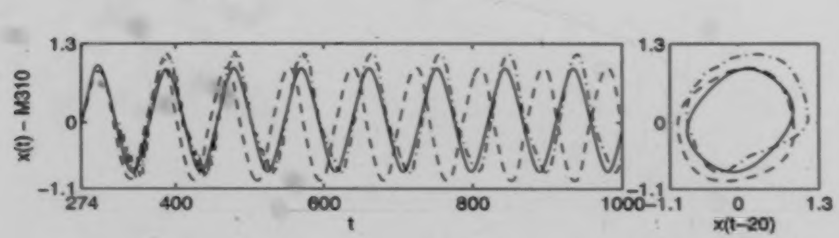

Figure 4.25: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M310.

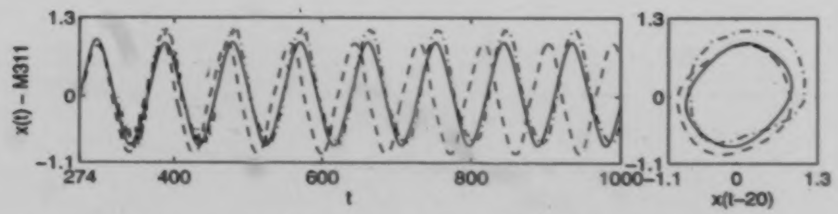

Figure 4.26: S1 ('—'), MSP for S1 ('- -'), and MSP for S2 ('-.'), for M311.

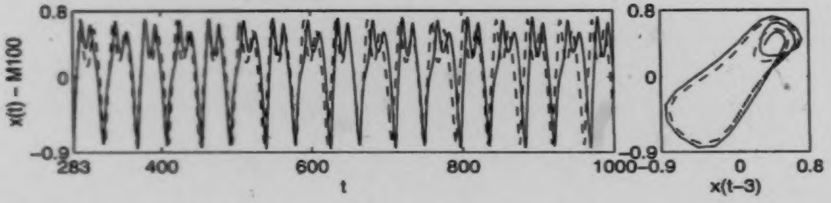

Figure 4.27: S3 ('—'), and MSP for S3 ('- -'), for M100. For S4, the ANN training did not converge.

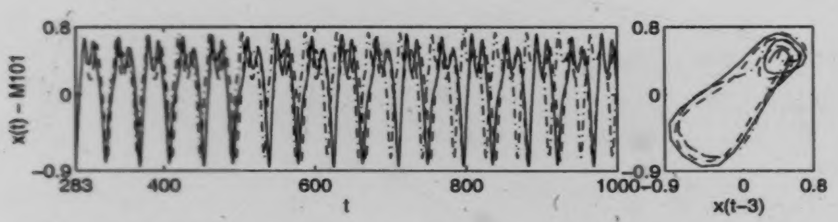

Figure 4.28: S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M101.

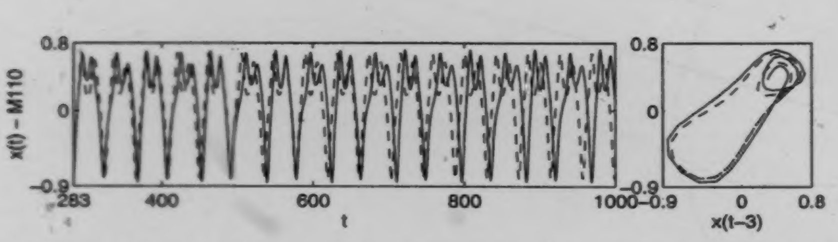

Figure 4.29: S3 ('—'), and MSP for S3 ('- -'), for M110. For S4, the ANN training did not converge.

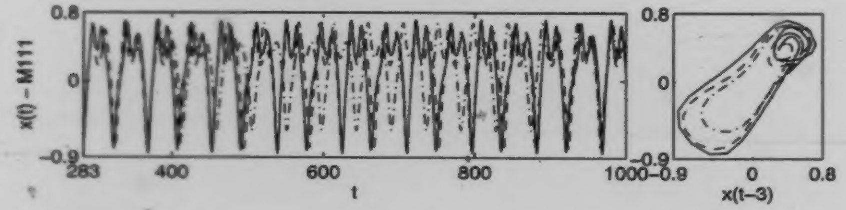

Figure 4.30: S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M111.

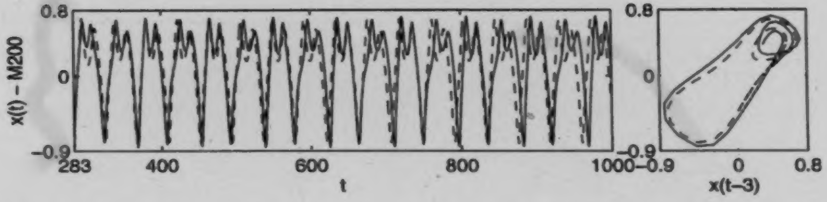

Figure 4.31: S3 ('—'), and MSP for S3 ('- -'), for M200. For S4, the ANN training did not converge.

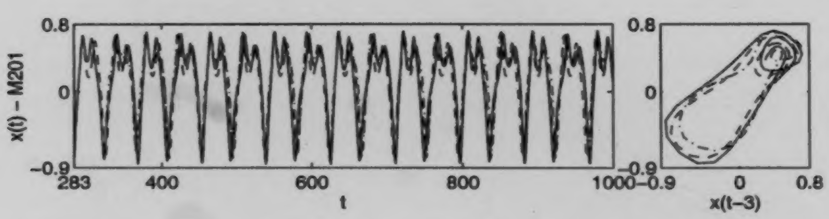

Figure 4.32: S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M201.

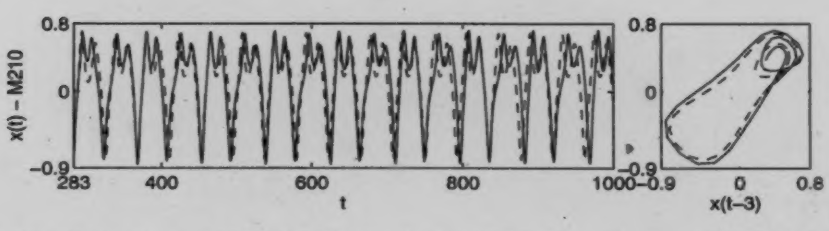

Figure 4.33: S3 ('—'), and MSP for S3 ('- -'), for M210. For S4, the ANN training did not converge.

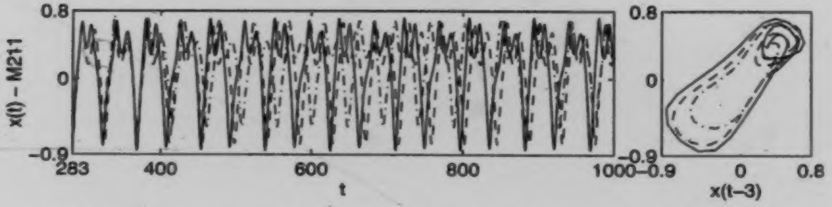

Figure 4.34: S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M211.

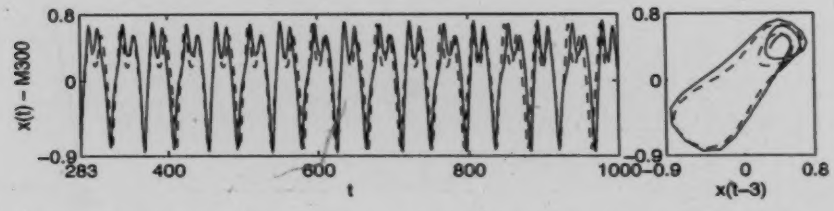

Figure 4.35: S3 ('—'), and MSP for S3 ('- -'), for M300. For S4, the ANN training did not converge.

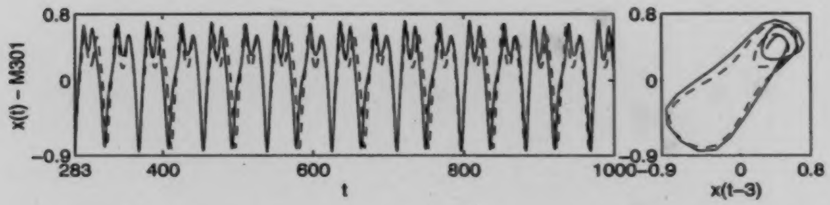

Figure 4.36: S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M301.

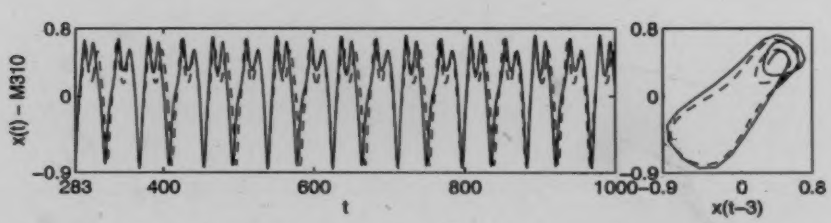

Figure 4.37: S3 ('—'), and MSP for S3 ('- -'), for M310. For S4, the ANN training did not converge.

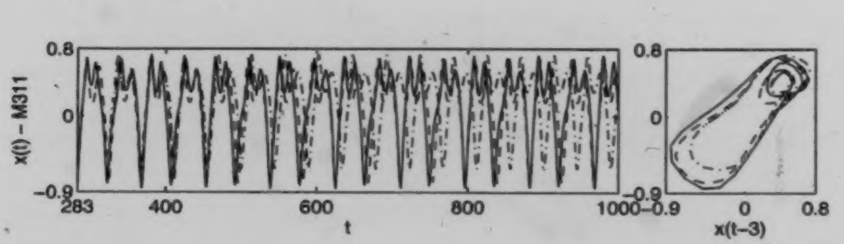

Figure 4.38: S3 ('—'), MSP for S3 ('- -'), and MSP for S4 ('-.'), for M311.

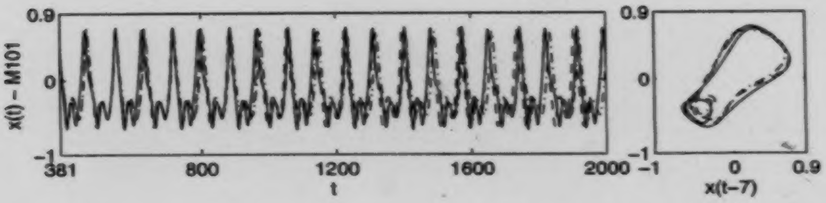

Figure 4.39: S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M101.

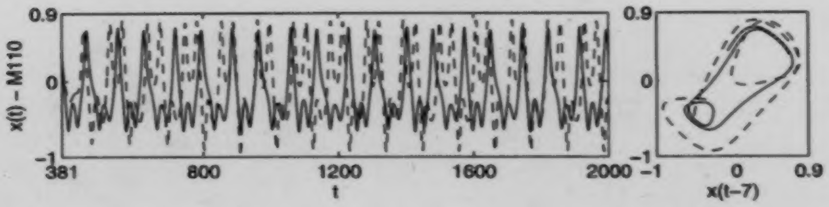

Figure 4.40: S5 ('—'), and MSP for S5 ('- -'), for M110. For S6, the ANN training did not converge.

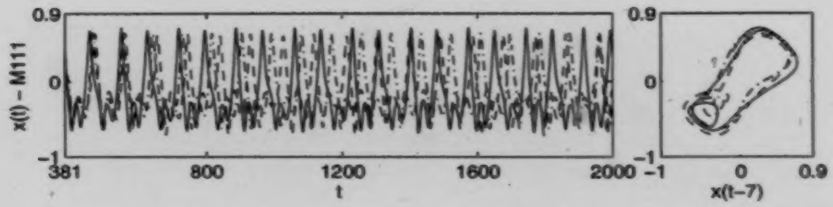

Figure 4.41: S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M111.

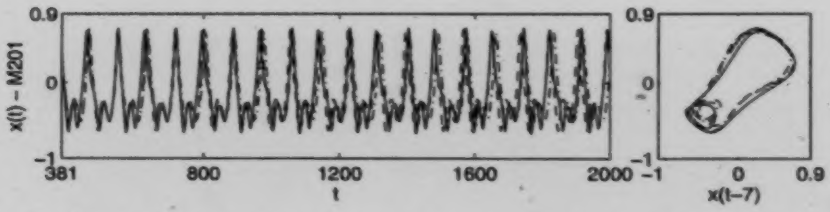

Figure 4.42: S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M201.

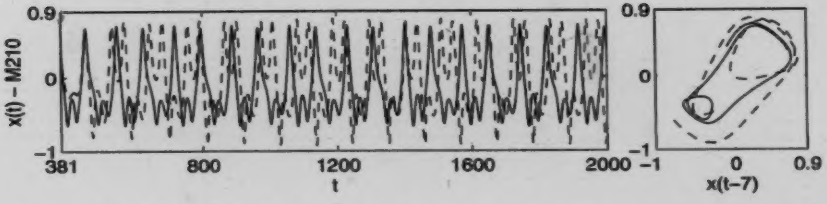

Figure 4.43: S5 ('—'), and MSP for S5 ('- -'), for M210. For S6, the ANN training did not converge.

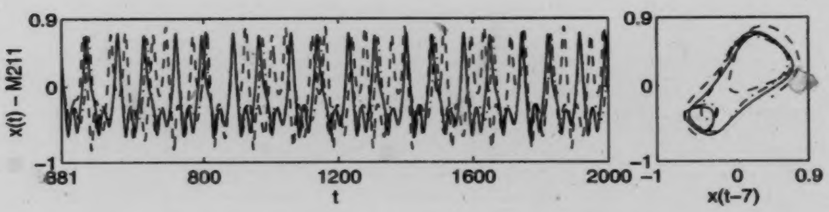

Figure 4.44: S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M211.

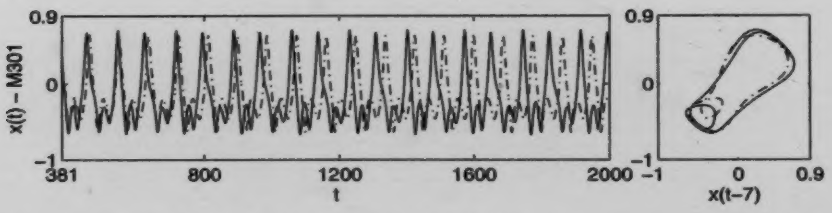

Figure 4.45: S5 ('—'), and MSP for S6 ('-.'), for M301. For S5, the ANN training did not converge.

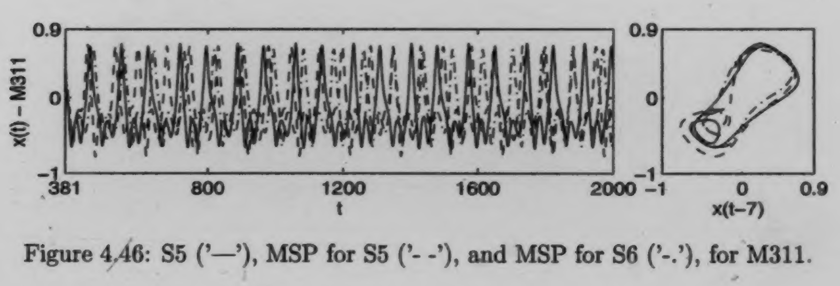

Figure 4.46: S5 ('—'), MSP for S5 ('- -'), and MSP for S6 ('-.'), for M311.

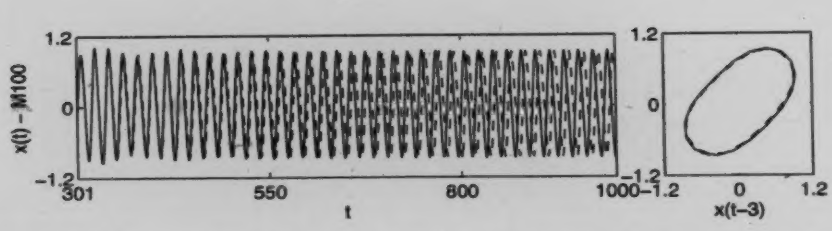

Figure 4.47: S7 ('—'), and MSP for S7 ('- -'), for M100. For S8, the ANN training did not converge.

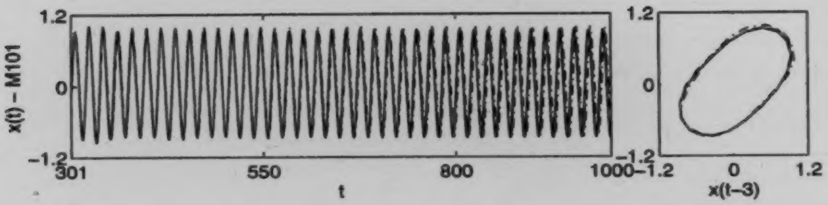

Figure 4.48: S7 ('—'), MSP for S7 ('- -'), and MSP for S8 ('-.'), for M101.

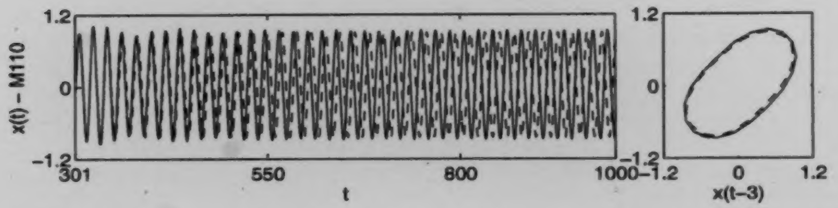

Figure 4.49: S7 ('—'), and MSP for S7 ('- -'), for M110. For S8, the ANN training did not converge.

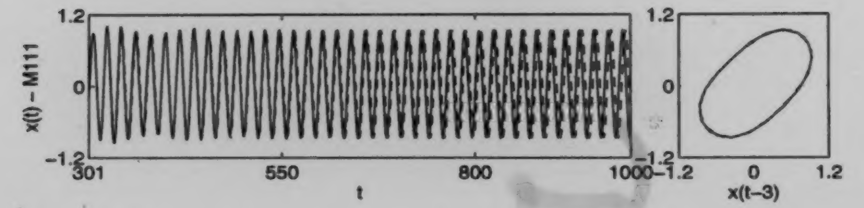

Figure 4.50: S7 ('—'), and MSP for S7 ('- -'), for M111. For S8, the ANN training did not converge.

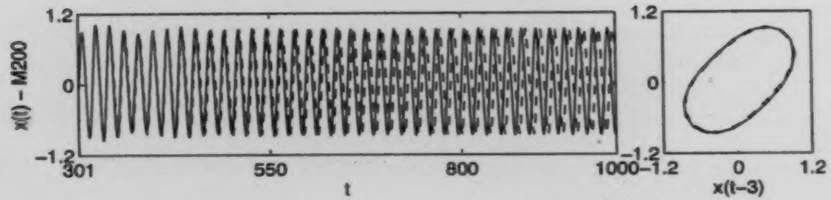

Figure 4.51: S7 ('—'), and MSP for S7 ('- -'), for M200. For S8, the ANN training did not converge.

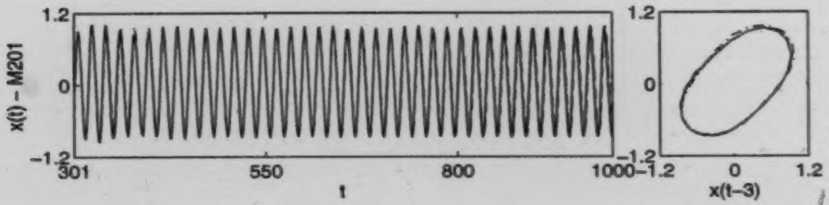

Figure 4.52: S7 ('—'), MSP for S7 ('- -'), and MSP for S8 ('-.'), for M201.

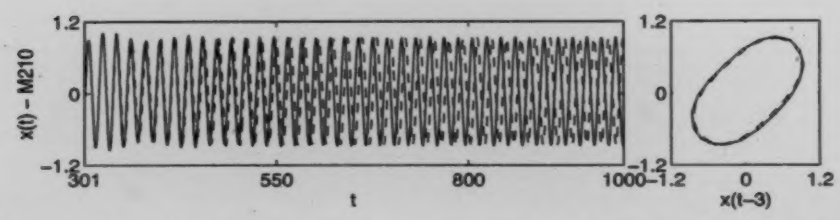

Figure 4.53: S7 ('—'), and MSP for S7 ('- -'), for M210. For S8, the ANN training did not converge.

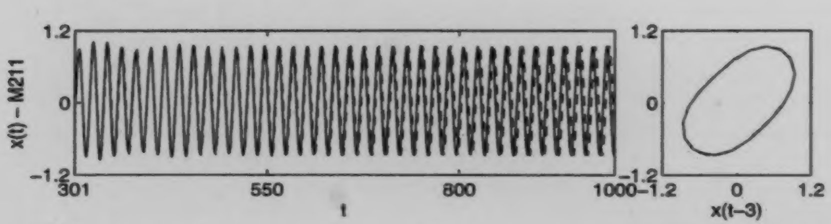

Figure 4.54: S7 ('—'), and MSP for S7 ('- -'), for M211. For S8, the ANN training did not converge.

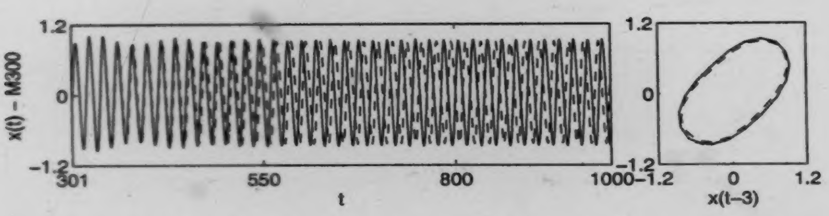

Figure 4.55: S7 ('—'), and MSP for S7 ('- -'), for M300. For S8, the ANN training did not converge.

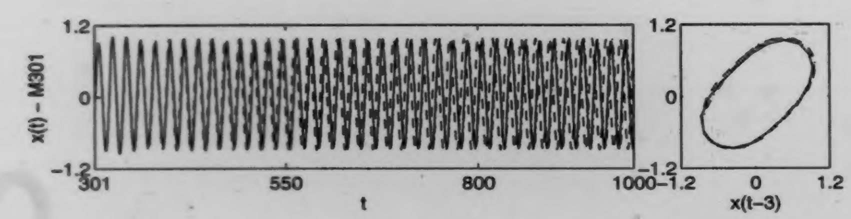

Figure 4.56: S7 ('—'), MSP for S7 ('- -'), and MSP for S8 ('-.'), for M301.

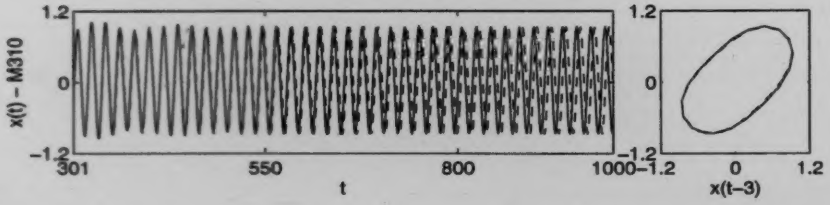

Figure 4.57: S7 ('—'), and MSP for S7 ('- -'), for M310. For S8, the ANN training did not converge.

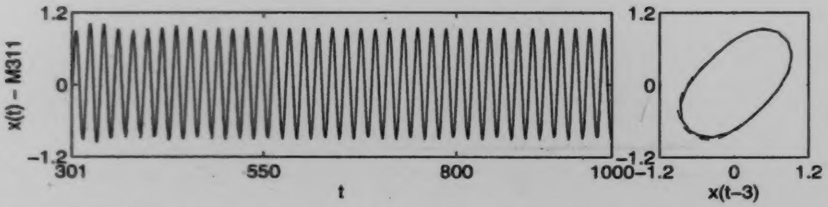

Figure 4.58: S7 ('—'), and MSP for S7 ('- -'), for M311. For S8, the ANN training did not converge.

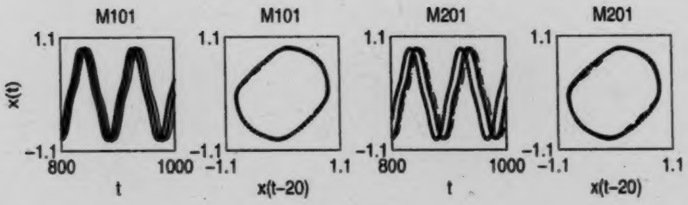

Figure 4.59: MSP of S1 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$('...'), 0.9('- -.'), 1.0('–'), 1.1('-.'), 1.2('- -').

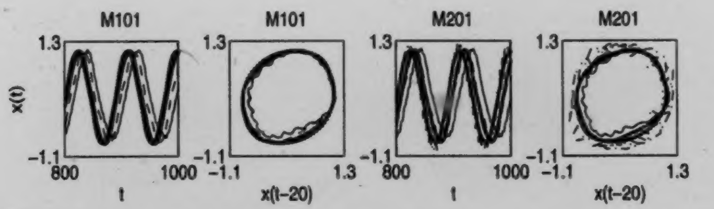

Figure 4.60: MSP of S2 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$('...'), 0.9('- -.'), 1.0('–'), 1.1('-.'), 1.2('- -'). For M101 with $\delta = 1.1$, the ANN training did not converge.

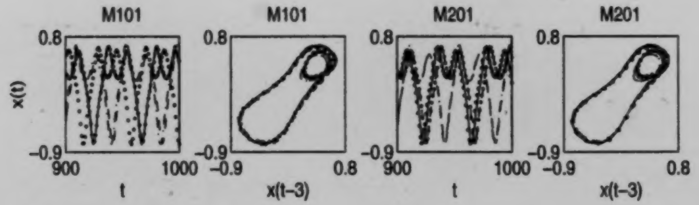

Figure 4.61: MSP of S3 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$('...'), 0.9('- -.'), 1.0('–'), 1.1('-.'), 1.2('- -').

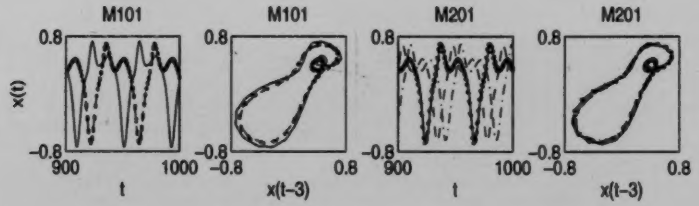

Figure 4.62: MSP of S4 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$('...'), 0.9('- -.'), 1.0('–'), 1.1('-.'), 1.2('- -'). For M101 with $\delta = 1.1$ and 1.2, the ANN training did not converge.

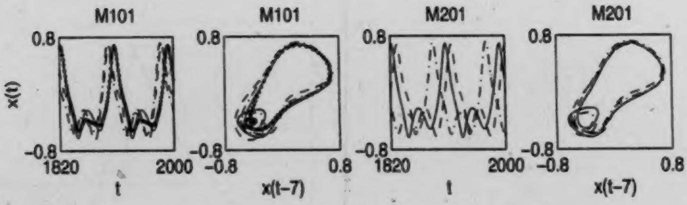

Figure 4.63: MSP of S5 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$('...'), 1.0('–'), 1.1('-.'), 1.2('- -'). For M101 with $\delta = 0.9$ and for M201 with $\delta = 0.8$ and 0.9, the ANN training did not converge.

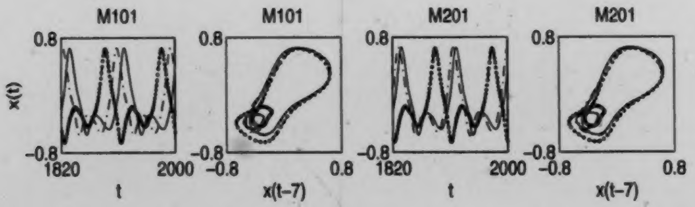

Figure 4.64: MSP of S6 for $n_0 = \delta \times n_0^*$, $\delta = 0.9$('- -.'), 1.0('–'), 1.1('-.') only.

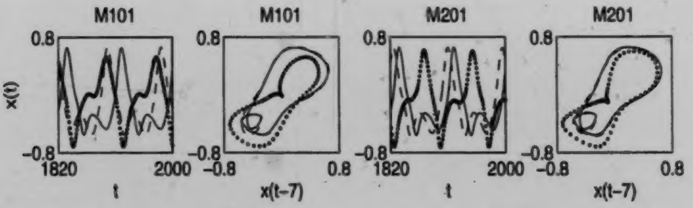

Figure 4.65: MSP of S6 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$('...'), 1.0('–'), 1.2('- -') only.

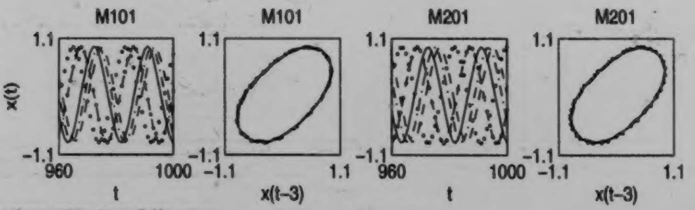

Figure 4.66: MSP of S7 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$('...'), 0.9('- -.'), 1.0('–'), 1.1('-.'), 1.2('- -').

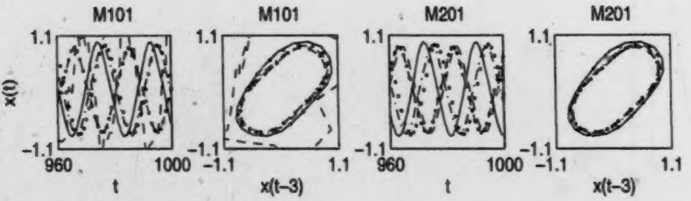

Figure 4.67: MSP of S8 for $n_0 = \delta \times n_0^*$, $\delta = 0.8$('...'), 0.9('- -.'), 1.0('–'), 1.1('-.'), 1.2('- -').

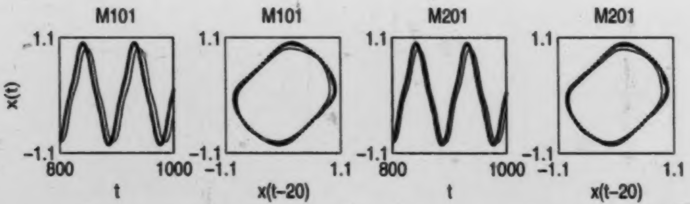

Figure 4.68: MSP of S1 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...').

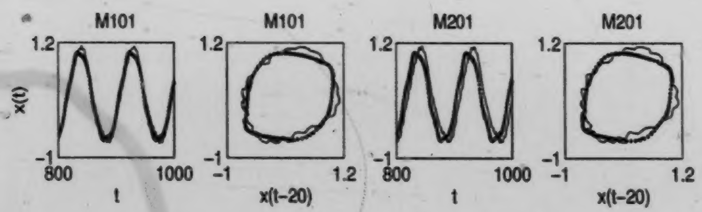

Figure 4.69: MSP of S2 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...').

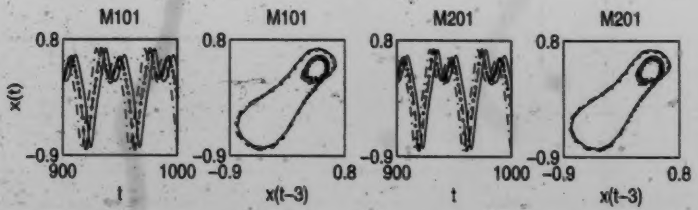

Figure 4.70: MSP of S3 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...').

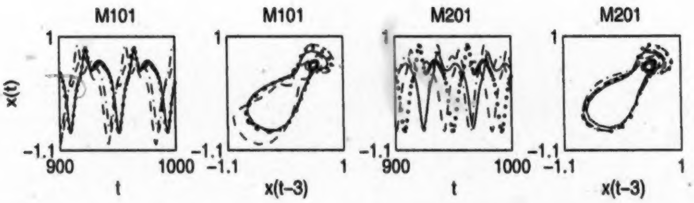

Figure 4.71: MSP of S4 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...').

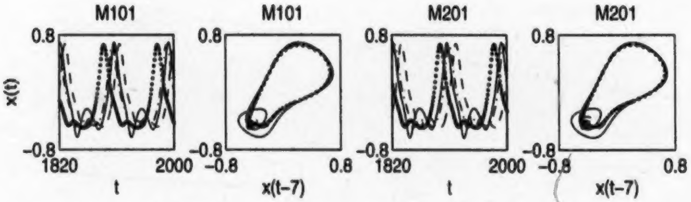

Figure 4.72: MSP of S5 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...').

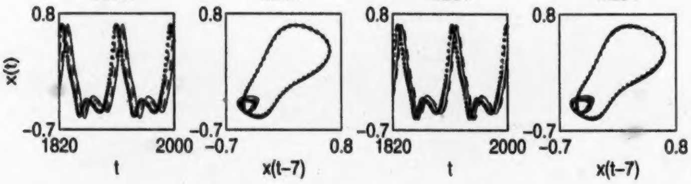

Figure 4.73: MSP of S6 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...').

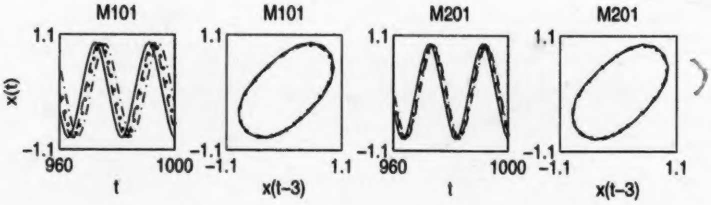

Figure 4.74: MSP of S7 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...').

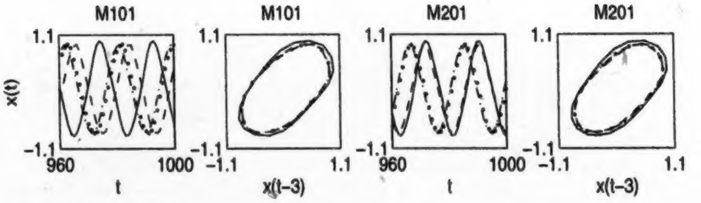

Figure 4.75: MSP of S8 for $n_1 = n_1^* + \delta$, $\delta = 0$('—'), 1('- -'), 2('-.'), 3('...').

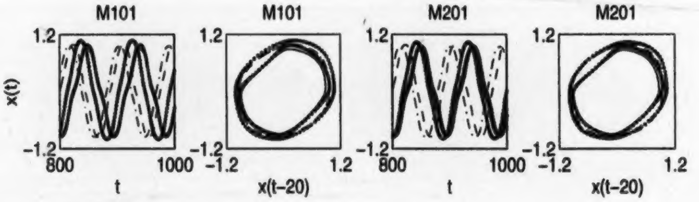

Figure 4.76: MSP of S1 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -').

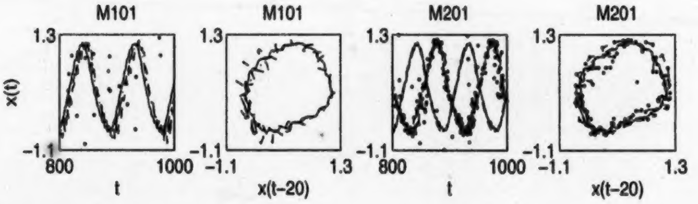

Figure 4.77: MSP of S2 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -'). For M101 with $\delta = -0.1$, the ANN training did not converge.

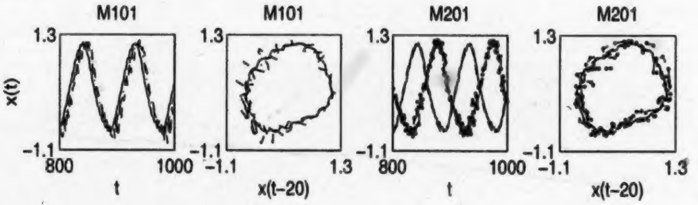

Figure 4.78: MSP of S2 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.1$('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -') only. For M101 with $\delta = -0.1$, the ANN training did not converge.

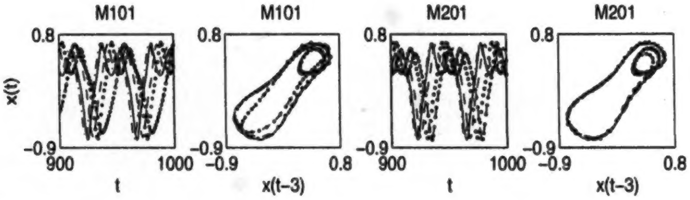

Figure 4.79: MSP of S3 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -').

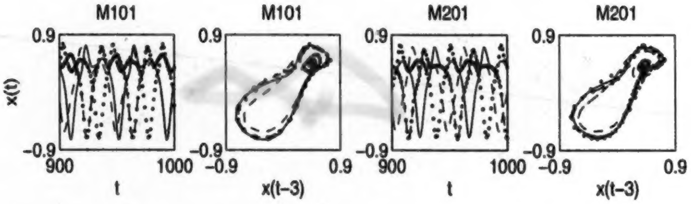

Figure 4.80: MSP of S4 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.2('- -'). For M101 and M201 with $\delta = +0.1$, the ANN training did not converge.

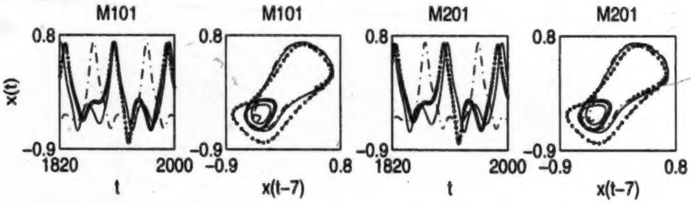

Figure 4.81: MSP of S5 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.1$('- -.'), 0.0('–'), 0.1('-.') only.

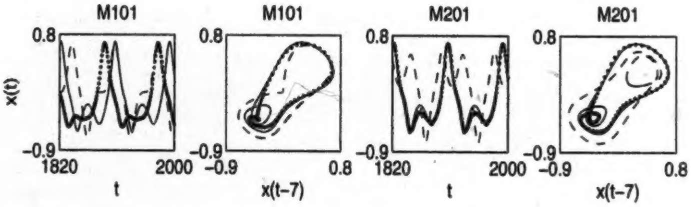

Figure 4.82: MSP of S5 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), 0.0('–'), 0.2('- -') only.

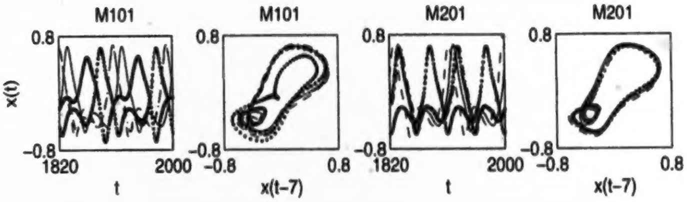

Figure 4.83: MSP of S6 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -').

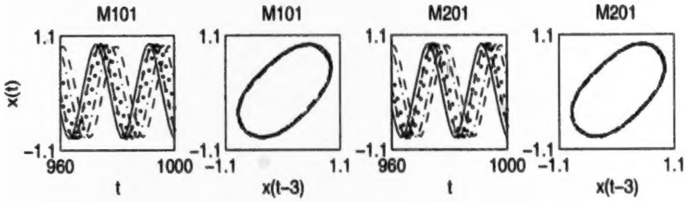

Figure 4.84: MSP of S7 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -').

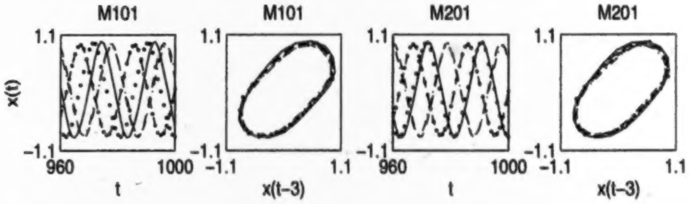

Figure 4.85: MSP of S8 for $t_1 = t_1^* + \delta \times n_0^*$, $\delta = -0.2$('...'), -0.1('- -.'), 0.0('–'), 0.1('-.'), 0.2('- -').

Chapter 5

Further Applications

5.1 Cavity Oscillations

Self-sustained oscillations over open cavities, such as weapons bays or automobile door gaps, have been extensively studied[1]. Recently there has been renewed interest in this type of phenomena, especially concerning the ability to control the cavity oscillations [154, 204]. The mathematical formulation for the cavity model is complicated, and it requires the solution of Navier-Stokes equations. The corresponding dynamics is also complex since it may involve interaction due to different modes in the system. In this section, we investigate the MSP capabilities for this difficult problem. Consider the test signals S9 and S10, obtained by selecting the first column of the data file M219D135.DAT and the 10-th column of the data file M219D085.DAT respectively. These signals were scaled by a factor of 0.1 in order to obtain an amplitude close to 1. The data files are in public domain, and they have been reported in NATO publication [273]. In the following, we will assume that the first $t_1 = 2000$ data points of both signals are known. These data sets will be used for ANN training and are shown in Fig.5.1.

Both signals S9 and S10 contain more than 20000 data points, and neither of the two trajectories converges to a limit cycle or a fixed point. The signals are bounded and have approximately constant mean and variance. S9 has a single dominant frequency while S10 exhibits three dominant frequencies whose relative energies vary with time. The FFT plots for the known data sets displayed in Fig.5.1 are provided in Figs.5.2-5.3, and are constructed by taking into account that both signals are sampled at a rate of 6 kHz. It is clear that there is one dominant mode at 492 Hz in S9 while the dynamics in S10 consists of three modes at 120 Hz, 360 Hz and 600 Hz respectively. The goal of our experiment will be to accurately reconstruct the dynamics corresponding to the dominant frequencies by performing a medium-term MSP using the M201 predictor presented in the previous chapters. More specifically, if n_0 is the chosen number of

---

[1] A version of part of the material in this section has been submitted for publication in [242].

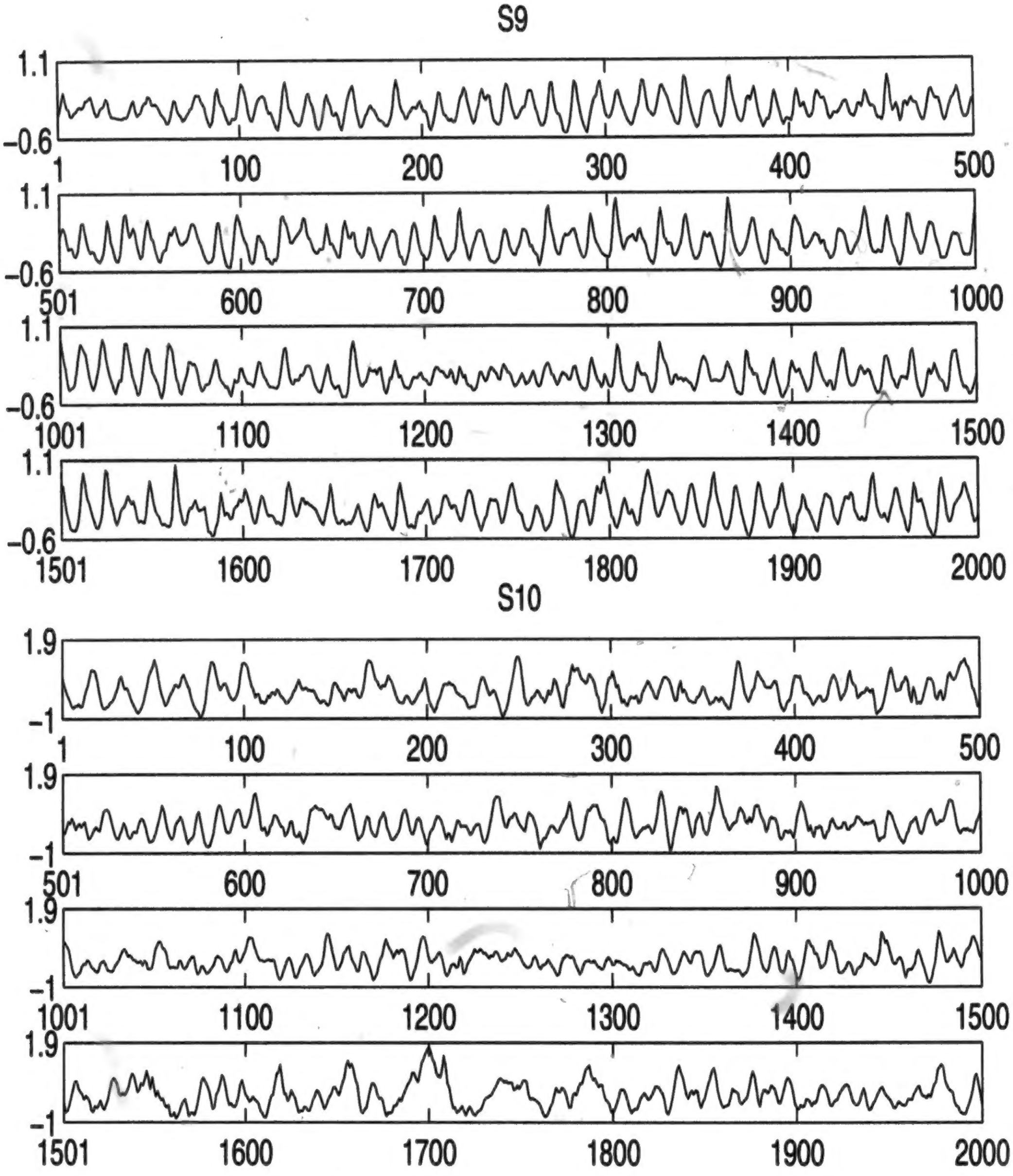

Figure 5.1: First 2000 data points of S9 and S10.

ANN inputs for a given signal, we will generate a data set of length $2n_0$ (i.e., $t_2 = t_1 + 2n_0$) by MSP.

The first step of our prediction method is to choose the number of ANN inputs n_0. The time

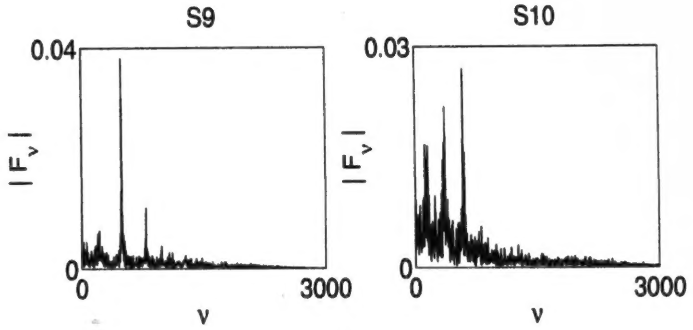

Figure 5.2: FFT for S9 and S10, based on the first 2000 data points.

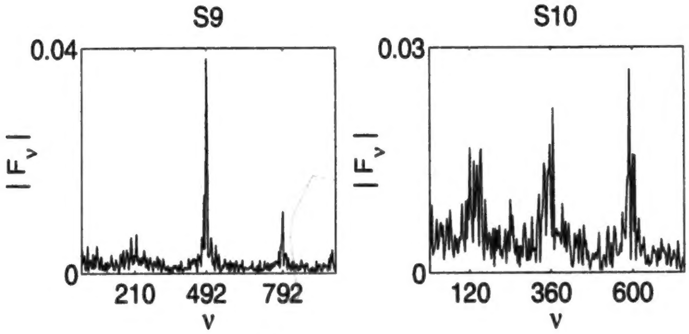

Figure 5.3: FFT (truncated plot) for S9 and S10, based on the first 2000 data points.

histories of the known data sets do not exhibit any clear periodicity, especially in the case of S10. The significance of an "oscillation cycle" is not clear in that case. For S9, most of the successive Lmax (the same for Lmin) points are situated at an approximately constant distance of 12 points from each other. However, due to the complicated profile of the signal, choosing $n_0 = 12$ is most likely not a good idea. Further insight into the periodicity properties of the two signals can be obtained by computing the ACF. The corresponding graphs are represented in Fig.5.4 for both signals. The vertical line marks the limit of reliability of the estimates for the ACF, as mentioned in Chapter 3 [68]. For both signals it has been noticed that each observation in the time series is correlated to all past observations, and the correlation is stronger for S9 than for S10. The ACF does not decay to zero as the lag increases. In the case of S10, the ACF becomes zero for $h \approx 250$, and subsequently increases again. The same happens for S9 at $h \approx 1100$, but this value is beyond the reliability bound for ACF estimates, hence it is not clear whether it should be used at all. Finally, from the FFT plots of the two training sets constructed based on a sampling step equal to unity (see Fig.5.5), estimates for the number of ANN inputs are given by $1.0/0.082 \approx 12$ and $1.0/0.02 \approx 50$ for S9 and S10 respectively.

The results presented in the following were obtained by using the method M201 with $n_1 = 2$

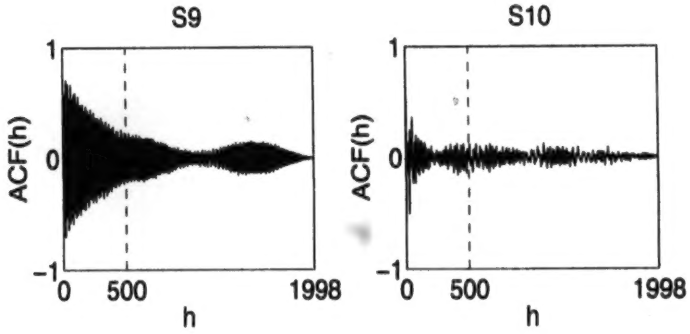

Figure 5.4: ACF for S9 and S10, based on the first 2000 data points.

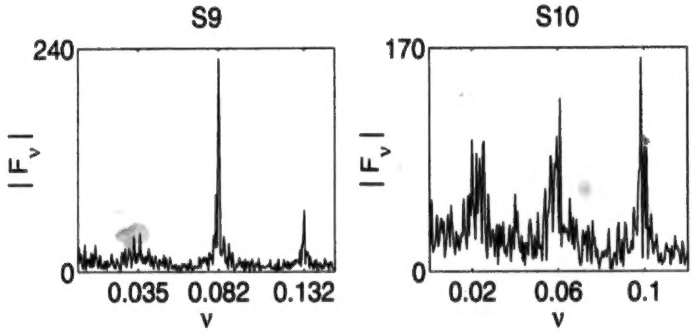

Figure 5.5: FFT (sampling step = 1) for S9 and S10, based on the first 2000 data points.

and $n_0 = 350$ (for S9), $n_0 = 250$ (for S10). The MSP generated after ANN training in each case is represented by the dashed line in Fig.5.6. Obviously, the prediction is not accurate if we compare the THs of the correct and the predicted signal. However, from further investigation of the MSP results, it has been revealed that the proposed ANN approach is capable of predicting the system dynamics, in particular correctly predicting the mode interactions.

For S9, the FFT spectra of the data sets $\{s(t_1+1),\ldots,s(t_1+n_0)\}$ and $\{\hat{y}_{\mathbf{w}}(t_1+1),\ldots,\hat{y}_{\mathbf{w}}(t_1+n_0)\}$ are compared, as well as the FFT spectra of the data sets $\{s(t_1+n_0+1),\ldots,s(t_1+2n_0)\}$ and $\{\hat{y}_{\mathbf{w}}(t_1+n_0+1),\ldots,\hat{y}_{\mathbf{w}}(t_1+2n_0)\}$. For S10, the spectra of the data sets $\{s(t_1+1),\ldots,s(t_1+2n_0)\}$ and $\{\hat{y}_{\mathbf{w}}(t_1+1),\ldots,\hat{y}_{\mathbf{w}}(t_1+2n_0)\}$ are compared. The spectrum of the correct and predicted signal as well as their overlapping in each case are plotted in Figs.5.7-5.9.

For S9, the dynamical system is essentially governed by a dominant mode at about 492 Hz. Using the 2000 data points as training set, although the predicted TH is not the same as the experimental signal as shown in Fig.5.6, the MSP has the same frequency as the correct signal. Moreover, the FFT plots displayed in Figs.5.7-5.8 demonstrate a good agreement for the frequency spectrum although the frequency has been shifted to about 497.14 Hz. The predicted profile for S10 as illustrated in Fig.5.6 seems to be unacceptable. However, by comparing the

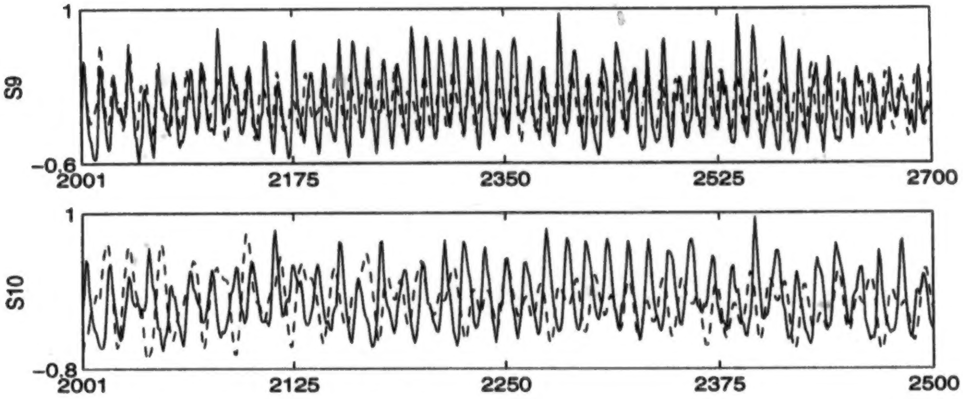

Figure 5.6: Correct signal ('—') and MSP ('- -') for S9 and S10.

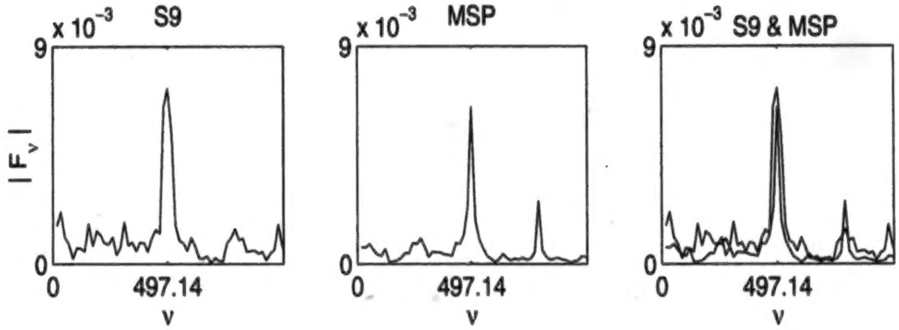

Figure 5.7: FFT for S9 and the ANN-MSP, based on the observations at the moments $t_1 + 1, \ldots, t_1 + n_0$.

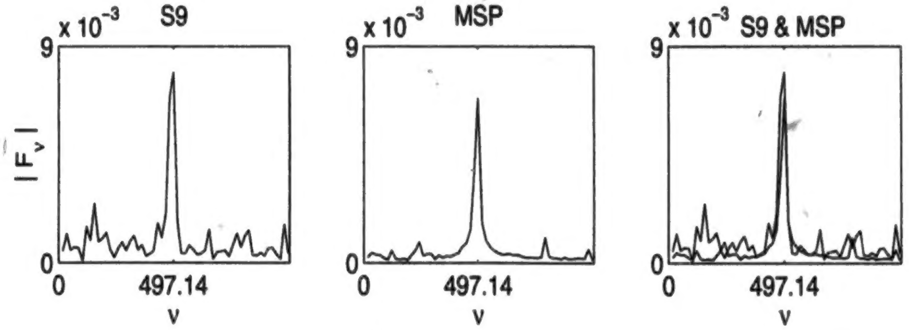

Figure 5.8: FFT for S9 and the ANN-MSP, based on the observations at the moments $t_1 + n_0 + 1, \ldots, t_1 + 2n_0$.

FFT plots for the predicted data and the experimnental data, shown in Fig.5.9, it is interesting to observe that the ANN MSP results correctly predict the dynamics of the system. In particular,

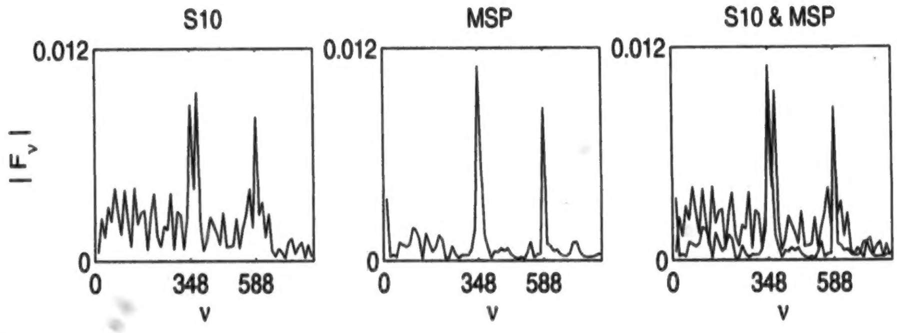

Figure 5.9: FFT for S10 and the ANN-MSP, based on the observations at the moments $t_1 + 1, \ldots, t_1 + 2n_0$.

the ANN is capable of forecasting the dynamics of mode interaction. Unlike the existence of three modes with the highest energy at 600 Hz as illustrated in Fig.5.3, the dynamics corresponding to that in Fig.5.9 has only two modes at 348 Hz and 588 Hz, and the largest amplitude is now located at 348 Hz.

Note that the results presented above are only preliminary results. Further work is necessary in order to improve the prediction accuracy for this highly complex problem. A preliminary preprocessing step, such as denoising and/or using a decomposition technique (for instance, the *empirical mode decomposition (EMD)*, proposed by Norden Huang ([100, 101]), followed by an ANN prediction of each component signal, might lead to more accurate results. However, as the above results demonstrate, the proposed forecasting method has great potential [illegible]edicting the dynamics of complex models.

5.2 Feature Extraction

In addition to performing LTMSP, ANNs can be trained to extract important features from the predicted signal[2]. Wong et.al. [257] proposed a combined wavelet-ANN model for extracting damping coefficients and modal frequency values of simulated signals. Suppose we are given a transient segment $\{\bar{s}(1),\ldots,\bar{s}(t_1)\}$, such as the 160 data points ($t_1 = 160$) marked by the vertical line in Fig.5.10, sampled with step 1.0/128 from a signal $s(\tau) = s_1(\tau) + s_2(\tau)$, $s_j(\tau) = A_j \exp(-a_j\tau)\sin(f_j\tau + \phi_j)$, $j = 1, 2$, where $A_1 = A_2 = 0.5$, $\phi_1 = \phi_2 = 0$, $a_1 = 0.7$, $a_2 = 0.5$, $f_1 = 12\pi$, $f_2 = 8\pi$, and $\phi_1 = \phi_2 = 0$, represent the amplitudes, damping ratios, modal frequencies and phase angle respectively, associated with the two Fourier modes.

Clearly, it is difficult to extract the damping coefficients and frequency values directly from such a short transient data set. In Fig.5.11 we present a wavelet-ANN model that can efficiently estimate the values of a_1, a_2, f_1, f_2, when the transient signal $\{\bar{s}(1),\ldots,\bar{s}(t_1)\}$ is provided. First,

---

[2] A version of the discussion presented in this section has been published in [238, 239].

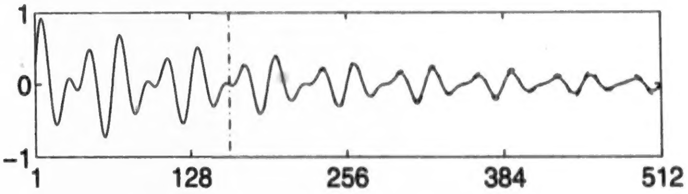

Figure 5.10: Simulated two-mode signal ('—') and ANN prediction ('- -').

an ANN as presented in the previous chapters (M201, with $n_0 = 110$, $n_1 = 2$) is trained using the transient data set and subsequently predicts a long-term nonlinear behavior $\{s(t); t \geq t_1 + 1\}$ (represented by the dashed line in Fig.5.10). Then $\{s(t); t \geq 1\}$ is fed into a wavelet decomposition module, where the two-mode signal is decomposed into two single-mode signals $\{s_1(t); t \geq 1\}$, $\{s_2(t); t \geq 1\}$. The details are described in [257]. A simple way to estimate the damping a and frequency f for a single-mode signal is to train a 2LFF1S2LNN to recognize the damping coefficients of damped sine waves $e^{-a\tau}\sin(f\tau)$ for (a, f) in some fixed bounded set $[a_i, a_f] \times [f_i, f_f]$. Then, given an arbitrary damped sine wave, its frequency f can be easily determined by using the FFT. If it is found that $f \in [f_i, f_f]$, then f and a sampling of the sine wave are provided as the ANN input. The corresponding network output will then provide an estimate for a.

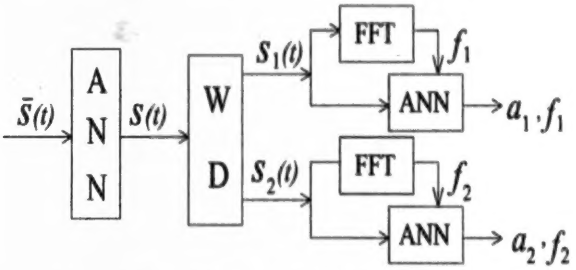

Figure 5.11: A Wavelet-ANN model for feature extraction.

In our experiments, a set of $Q = 100$ training pairs $\{(\tilde{a}^{(q)}, \tilde{f}^{(q)}); 1 \leq q \leq Q\}$, and a set of Q testing pairs $\{(\hat{a}^{(q)}, \hat{f}^{(q)}); 1 \leq q \leq Q\}$, uniformly distributed in $[0.1, 4.1] \times [3.0, 9.0]$, were simultaneously generated. The damped sine wave $e^{-a\tau}\sin(f\tau)$, $\tau \geq 0$ corresponding to each pair was sampled with step $\Delta t = 1.0/128$, generating $t_2 = 512$ discrete points in each case: $\{\tilde{s}^{(q)}(t); 1 \leq t \leq t_2\}$ and $\{\hat{s}^{(q)}(t); 1 \leq t \leq t_2\}$ respectively.

The 2LFFNN was trained to output an estimated value y for the damping coefficient a when receiving as inputs the value of f and the 512 sample points of the given single-mode signal. At each training iteration, for every $q = 1, 2, \ldots, Q$, the ANN receives the inputs $\tilde{f}^{(q)}$, $\tilde{s}^{(q)}(1)$, …, $\tilde{s}^{(q)}(t_2)$, and the network output $\tilde{y}^{(q)}$ approximates the correct output $\tilde{a}^{(q)}$ (the corresponding damping coefficient). Thus, the ANN had 513 inputs and one output. In the network implementation, 3 neurons were used in the hidden layer. The accuracy of the estimated value for a was measured by the relative output error $|(a-y)/a|$. The network training consisted in applying the conjugate gradient algorithm to minimize the mean squared relative output error over the $Q = 100$ training patterns (the performance index E_R) as a function of the 1546 network weights:

$$E_R(\mathbf{w}) = \frac{1}{Q}\sum_{q=1}^{Q}\left[\frac{\tilde{a}^{(q)} - \tilde{y}^{(q)}(\mathbf{w})}{\tilde{a}^{(q)}}\right]^2. \tag{5.1}$$

The training was stopped when $\sqrt{E_R(\mathbf{w})} < 0.015$. In the testing phase, the relative output errors $\left|\left(\hat{a}^{(q)} - \hat{y}^{(q)}(\mathbf{w})\right)/\hat{a}^{(q)}\right|$, $1 \le q \le Q$, are computed. The stopping criterion was reached after 1422 iterations (4 hours on a Sun workstation, Ultra-10 model). Even though the network gave responses within 5% accuracy for all training data, 12% of the network responses in the testing phase were associated with large errors in the range of over 5% to 30% (see Fig.5.12(a),(c)). In order to achieve a better testing performance, the training must be continued until $\sqrt{E_R(\mathbf{w})} \ll 0.015$.

A more efficient procedure for estimating a has been proposed, in which the network inputs consist of the value of $f^{(q)}$ and the first eight successive local maximum and minimum values of the corresponding signal: $\tilde{M}_1^{(q)}$, $\tilde{m}_1^{(q)}$, $\tilde{M}_2^{(q)}$, $\tilde{m}_2^{(q)}$, $\tilde{M}_3^{(q)}$, $\tilde{m}_3^{(q)}$, $\tilde{M}_4^{(q)}$, $\tilde{m}_4^{(q)}$ (4 local maximum and 4 local minimum values) as inputs. The neural network now has only nine inputs, three neurons in the hidden layer, and one output, and this reduces the number of weights to 34. The value of $\tilde{f}^{(q)}$ has to be included in the network input, since the local maximum and minimum values by themselves

$$\frac{(-1)^k \tilde{\eta}^{(q)}}{\sqrt{1+\left(\tilde{\eta}^{(q)}\right)^2}} \exp\left\{\frac{k\pi - \arctan\left(\tilde{\eta}^{(q)}\right)}{\tilde{\eta}^{(q)}}\right\} \tag{5.2}$$

($k \in \mathbf{Z}$) only determine the quotient $\tilde{\eta}^{(q)} = \tilde{f}^{(q)}/\tilde{a}^{(q)}$.

In this case, the network training stopped after 1440 iterations, and the output errors for the training set were all within 5%, as shown in Fig.5.12(b). The training process took only seven minutes, and more accurate estimates for the damping values were obtained. As shown in Fig.5.12(d), 98% of the estimates were within 5% accuracy and only 2% had error in the range of 10% to 15%. The result represents a significant improvement in both training time and output accuracy. To obtain all testing errors within 5%, additional training is required.

Clearly, the present wavelet-ANN approach can be extended to a general n-mode signal. Our damping extraction method is an alternative approach to the one proposed by Johnson et.al. [110], who applied the discrete wavelet transform on both the given signal and a dictionary of so-called "singlet functions" and performed correlation filtering to determine which singlet function best approximates the frequency and damping characteristics of the given signal at a specific point in time. In our case, a dictionary of artificial neural networks trained to output the damping of a single-mode signal for different frequency intervals could be built and used together with wavelet decomposition to provide real-time estimates of damping coefficients.

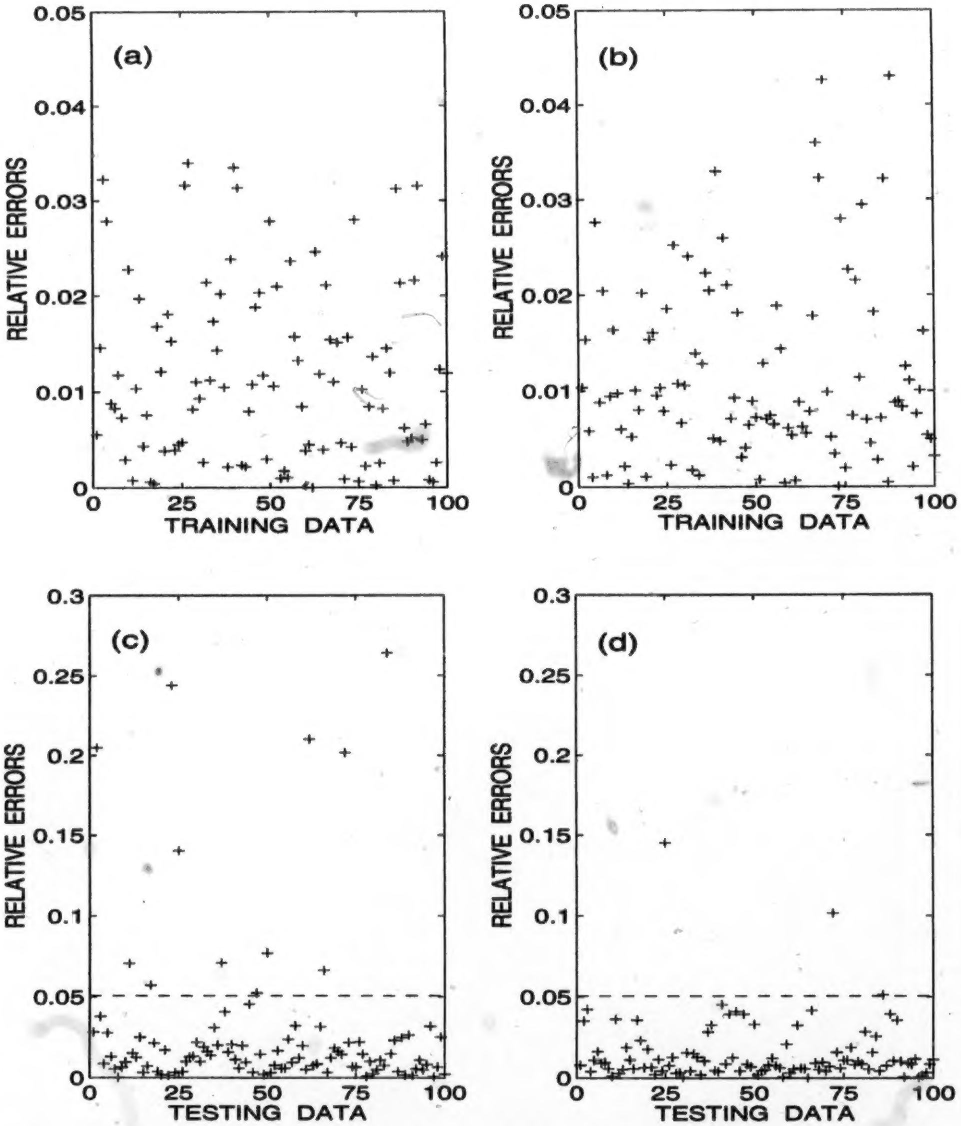

Figure 5.12: Training ((a),(b)) and testing ((c),(d)) relative errors when using 513 ((a),(c)) and 9 ((b),(d)) network inputs respectively, in the damping extraction problem.

Chapter 6

Conclusions and Future Directions

In this thesis, a new approach based on the use of artificial neural networks (ANNs) for long-term prediction of nonlinear oscillations arising from dynamical systems was proposed. An ANN is trained using a segment of the transient state of a signal, and the asymptotic state of the trajectory is reconstructed by a multi-step prediction (MSP) process. The problem of a long-term MSP using ANNs has not yet been thoroughly investigated, since most of the ANN research deals mainly with interpolative problems. For an accurate MSP, it is essential to construct a stable nonlinear mapping and to overcome the problem of error propagation, especially given the fact that the known data set is always corrupted by measurement noise. In fact, the present work points to a novel research direction in the field of ANNs, for which new tools need to be created in order to deal with aspects not encountered in interpolative applications.

In this study, an original MSP method using ANNs was designed, with special features that control the propagation of the prediction errors. The proposed predictor was tested on numerically generated signals and real-life experimental data. Based on a detailed comparison of 12 combinations of network architectures and training algorithms, it was found that the method that extracts the maximum amount of information from the training set using the minimum number of neurons, while providing the greatest robustness in the presence of signal noise and variations in the number of inputs, neurons, and in the endpoint of the training set, is the one using a two-layer feedforward ANN with normalized second layer weights, trained with a constant learning rate, and for which the first layer weights are initialized with normalized segments of the training signal. Guidelines for consistently choosing the number of network inputs and hidden neurons have also been reported.

A theoretical justification of the proposed approach is presented, showing that, under certain conditions, the ω-limit sets of the original trajectory and of the predicted signal are close to each other. The theoretical analysis could be further refined by providing an estimate for the mapping $\epsilon \mapsto \tilde{b}_{\mathbf{w}}(\epsilon)$ in Section 3.4, which depends only on the form of $\mathbf{\Phi}_{\mathbf{w}}$ and on the values of the ANN weights. This could constitute a subject for further extensive theoretical research and would lead to the possibility of providing a practically relevant bound on the accuracy of the ANN-generated prediction, that would be available at the end of network training.

As a further application of the proposed prediction approach, we demonstrate that the developed ANN is capable of predicting the nonlinear dynamics for the complex cavity flows. More work is needed in order to refine the prediction, in particular to capture the dynamics in multi-mode interactions. A possible approach could be to decompose the complex nonlinear signal into a sum of nonlinear components (the *empirical mode decomposition (EMD)*, proposed by Norden Huang [100, 101]), and to perform LTMSP of each component separately. The cavity flow also leads to the general issue of finding better measures for the prediction accuracy. As already mentioned, the TH-based pointwise error is not a relevant measure since it could be large when the profile of the predicted signal is accurate but there is a phase shift between the MSP and the correct signal. An option worth investigating could be the following. For each lag d, $1 \leq d \leq n_0$, plot $s(t)$ vs. $s(t-d)$ and $\hat{y}_{\mathbf{w}}(t)$ vs. $\hat{y}_{\mathbf{w}}(t-d)$ for the last oscillation cycle $(t_2 - n_0 + 1 \leq t \leq t_2)$. Interpolate between the consecutive points $\big(s(t-d-1), s(t-1)\big)$ and $\big(s(t-d), s(t)\big)$ (for example, by segments), obtaining a curve $\gamma_s : [t_2 - n_0 + 1, t_2] \to \mathbf{R}^2$. Repeat the process for all points $\big(\hat{y}_{\mathbf{w}}(t-d-1), \hat{y}_{\mathbf{w}}(t-1)\big)$ and $\big(\hat{y}_{\mathbf{w}}(t-d), \hat{y}_{\mathbf{w}}(t)\big)$, obtaining another curve $\gamma_{\hat{y}_{\mathbf{w}}} : [t_2 - n_0 + 1, t_2] \to \mathbf{R}^2$. A scalar measure of the area (in the plane) between the two curves could be defined in the form of an integral, in a similar manner to the area bounded by the graphs of two functions $f, g : [a, b] \to \mathbf{R}$ (for some $a < b$), defined as $\int_a^b |f(\xi) - g(\xi)|\, d\xi$. However, the generalization of this formula to arbitrary, self-intersecting planar curves is not straightforward.

An ANN can be trained to extract the values of the damping coefficients from the predicted signal, when used in conjunction with a wavelet decomposition technique. A feature extraction method was designed, which dramatically improves the network training time when extracting the damping coefficient from a single-mode signal. Further work needs to be done in order to extend the damping extraction method to multi-mode signals without using a wavelet decomposition module.

In a parallel study at the University of Alberta, nonlinear time series models and the extended Kalman filter have also been used for providing long-term MSP of nonlinear oscillations [194, 258, 259, 260]. The training sets necessary for those methods in order to provide an accurate MSP are about 25% smaller than those necessary using the ANN-based predictors. In nonlinear time series models, the model parameters are estimated more sharply than in the case of ANNs. This, however, also causes these methods to be sensitive to signal noise. In the case of a noisy signal, a denoising step is required before applying nonlinear time series models for long-term MSP. In using the extended Kalman filter, the form of the nonlinearity must be known beforehand. However, this information is usually not available in practical situations. Neural networks are robust with respect to noise and to variations in the parameters such as the number of inputs or neurons, or the values of the weights, due to the parallel and distributed storage and processing of information inside the ANNs. Thus, the parameters of the method do not need to be estimated sharply. Moreover, no additional information about the underlying dynamics needs to be provided to the ANNs. The shortcoming is that the ANNs need more training data than the nonlinear time series models in order to accurately predict the same signal. In addition, time series models have the advantage of being better understood and theoretically tractable than ANNs, even though the understanding is still incomplete for most nonlinear models. In a real-life application, all these three methods (nonlinear time series models, extended Kalman filter, and neural networks) could be used to provide a LTMSP. By comparing the MSP results, one could have a reasonable degree of confidence whether the prediction is correct.

The main assumption when using the proposed prediction approach is that the underlying dynamics of the given trajectory can be modeled by a finite-dimensional dynamical system, or at least can be accurately approximated by a finite-dimensional dynamics. Moreover, it is assumed that the training data set contains sufficient information of the nonlinear dynamical system. This constitutes the main limitation of the approach proposed in this study. The extension of this method to the prediction of infinite-dymensional dynamics is not straightforward. Further research needs to be done in order to deal with this type of problems. In general, even though the results obtained in this study can be further refined, we conclude that ANNs prove to be useful tools in long-term prediction of nonlinear trajectories and in feature extraction.

Bibliography

[1] O. Adetona, E. Garcia, and L. H. Keel, *A New Method for the Control of Discrete Nonlinear Dynamic Systems Using Neural Networks*, IEEE Trans. Neural Networks, Vol. 11, No. 1, 2000, pp. 102-112.

[2] D. F. Akhmetov, Y. Dote, and S. J. Ovaska, *Fuzzy Neural Network with General Parameter Adaptation for Modeling of Nonlinear Time Series*, IEEE Trans. Neural Networks, Vol. 12, No. 1, 2001, pp. 148-152.

[3] A. Alessandri, M. Sanguineti, and M. Maggiore, *Optimization-Based Learning with Bounded Error for Feedforward Neural Networks*, IEEE Trans. Neural Networks, Vol. 13, No. 2, 2002, pp. 261-272.

[4] S. I. Amari, N. Murata, K. R. Müller, M. Finke, and H. H. Yang, *Asymptotical Statistical Theory of Overtraining and Cross-Validation*, IEEE Trans. Neural Networks, Vol. 8, No. 5, 1997, pp. 985-995.

[5] N. Ampazis, S. J. Perantonis, and J. G. Taylor, *Dynamics of Multilayer Networks in the Vicinity of Temporary Minima*, Neural Networks, Vol. 12, 1999, pp. 43-58.

[6] R. Anand, K. Mehrorta, C. K. Mohan, S. Ranka, *Efficient Classification for Multiclass Problems Using Modular Neural Networks*, IEEE Trans. Neural Networks, Vol. 6, No. 1, 1995, pp. 117-124.

[7] A. F. Atiya, and A. G. Parlos, *New Results on Recurrent Network Training: Unifying the Algorithms and Accelerating Convergence*, IEEE Trans. Neural Networks, Vol. 11, No. 3, 2000, pp. 697-709.

[8] A. F. Atiya, S. M. El-Shoura, S. I. Shaheen, and M. S. El-Sherif, *A Comparison Between Neural-Network Forecasting Techniques - Case Study: River Flow Forecasting*, IEEE Trans. Neural Networks, Vol. 10, No. 2, 1999, pp. 402-409.

[9] A. Aussem, *Dynamical Recurrent Neural Networks Toward Prediction and Modeling of Dynamical Systems*, Neurocomputing 28 (1999), pp. 207-232.

[10] J. BALTERSEE, AND J. A. CHAMBERS, *Nonlinear Adaptive Prediction of Speech with a Pipelined Recurrent Neural Network*, IEEE Trans. Signal Processing, Vol. 46, No. 8, 1998, pp. 2207-2216.

[11] E. BARNARD, *Optimization for Training Neural Nets*, IEEE Trans. Neural Networks, Vol. 3, No. 2, 1992, pp. 232-240.

[12] R. BATTITI, AND G. TECCHIOLLI, *Training Neural Nets with the Reactive Tabu Search*, IEEE Trans. Neural Networks, Vol. 6, No. 5, 1995, pp. 1185-1200.

[13] Y. BENGIO, P. SIMARD, AND P. FRASCONI, *Learning Long-Term Dependencies with Gradient Descent is Difficult*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 240-253.

[14] V. L. BERARDI, AND G. P. ZHANG, *An Empirical Investigation of Bias and Variance in Time Series Forecasting: Modeling Considerations and Error Evaluations*, IEEE Trans. Neural Networks, Vol. 14, No. 3, 2003, pp. 668-679.

[15] C. S. BERGER, *Recursive Single-Layer Nets for Output Error Dynamic Models*, IEEE Trans. Neural Networks, Vol. 6, No. 2, 1995, pp. 508-511.

[16] F. BERNELLI-ZAZZERA, P. MANTEGAZZA, G. MAZZONI, AND M. RENDINA, *Active Flutter Suppression Using Recurrent Neural Networks*, Journal of Guidance, Control, and Dynamics, Vol. 23, No. 6, 2000, pp. 1030-1036.

[17] S. BHARITKAR, AND J. M. MENDEL, *The Hysteretic Hopfield Neural Network*, IEEE Trans. Neural Networks, Vol. 11, No. 4, 2000, pp. 879-888.

[18] M. BIANCHINI, M. GORI, AND M. MAGGINI, *On The Problem of Local Minima in Recurrent Neural Networks*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 167-177.

[19] S. A. BILLINGS, AND X. HONG, *Dual-Orthogonal Radial Basis Function Networks for Nonlinear Time Series Prediction*, Neural Networks 11 (1998), pp. 479-493.

[20] C. M. BISHOP, *Curvature-Driven Smoothing: A Learning Algorithm for Feedforward Networks*, IEEE Trans. Neural Networks, Vol. 4, No. 5, 1993, pp. 882-884.

[21] J. J. BLOCK, AND T. W. STRGANAC, *Applied Active Control for a Nonlinear Aeroelastic Structure*, Journal of Guidance, Control, and Dynamics, Vol. 21, No. 6, 1998, pp. 838-845.

[22] M. BRIN, AND G. STUCK, *Introduction to Dynamical Systems*, Cambridge University Press, Cambridge, 2002.

[23] M. BURGER, AND H. W. ENGL, *Training Neural Networks with Noisy Data as an Ill-Posed Problem*, Advances in Computational Mathematics, Vol. 13, 2000, pp. 335-354.

[24] D. BURSHTEIN, *Long-Term Attraction in Higher-Order Neural Networks*, IEEE Trans. Neural Networks, Vol. 9, No. 1, 1998, pp. 42-50.

[25] L. CAO, *Support Vector Machines Experts for Time Series Forecasting*, Neurocomputing, Vol. 51, 2003, pp. 321-339.

[26] G. CASTELLANO, A. M. FANELLI, AND M. PELILLO, *An Iterative Pruning Algorithm for Feedforward Neural Networks*, IEEE Trans. Neural Networks, Vol. 9, No. 1, 1998, pp. 42-50.

[27] J. L. CASTRO, C. J. MANTAS, AND J. M. BENÍTEZ, *Neural Networks with a Continuous Squashing Function in the Output are Universal Approximators*, Neural Networks, Vol. 13, 2000, pp. 561-563.

[28] H. A. CECCATO, H. D. NAVONE, AND H. WAELBROEK, *Learning Persistent Dynamics with Neural Networks*, Neural Networks, Vol. 11, No. 1, 1998, pp. 145-151.

[29] D. S. CHEN, AND R. C. JAIN, *A Robust Back Propagation Learning Algorithm for Function Approximation*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 1994, pp. 467-479.

[30] R. CHEN, *A Nonparametric Multi-Step Prediction Estimator in Markovian Structures*, Statistica Sinica, No. 6, 1996, pp. 603-615.

[31] T. CHEN, *A Unified Approach for Neural Network-like Approximation of Non-Linear Functionals*, Neural Networks, Vol. 11, 1998, pp. 981-983.

[32] T. CHEN, AND H. CHEN, *Approximation of Continuous Functionals by Neural Networks with Application to Dynamic Systems*, IEEE Trans. Neural Networks, Vol. 4, No. 6, 1993, pp. 910-918.

[33] T. CHEN, AND H. CHEN, *Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems*, IEEE Trans. Neural Networks, Vol. 6, No. 4, 1995, pp. 911-917.

[34] T. CHEN, H. CHEN, AND R. W. LIU, *Approximation Capability in* $C(\mathbf{R}^n)$ *by Multilayer Feedforward Networks and Related Problems*, IEEE Trans. Neural Networks, Vol. 6, No. 1, 1995, pp. 25-30.

[35] V. CHERKASSKY, D. GEHRING, AND F. MULIER, *Comparison of Adaptive Methods for Function Estimation from Samples*, IEEE Trans. Neural Networks, Vol. 7, No. 4, 1996, pp. 969-984.

[36] Y. M. CHEUNG, AND L. XU, *Independent Component Ordering in ICA Time Series Analysis*, Neurocomputing, Vol. 41, 2001, pp. 145-152.

[37] C. C. CHIANG, AND H. C. FU, *Using Multithreshold Quadratic Sigmoidal Neurons to Improve Classification Capability of Multilayer Perceptrons*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 1994, pp. 516-519.

[38] E. S. CHNG, S. CHEN, AND B. MULGREW, *Gradient Radial Basis Function Networks for Nonlinear and Nonstationary Time Series Prediction*, IEEE Trans. Neural Networks, Vol. 7, No. 1, 1996, pp. 190-194.

[39] S. Y. CHO, AND T. W. S. CHOW, *Learning Parametric Specular Reflectance Model by Radial Basis Function Network*, IEEE Trans. Neural Networks, Vol. 11, No. 6, 2000, pp. 1498-1503.

[40] C. H. CHOI, AND J. Y. CHOI, *Constructive Neural Networks with Piecewise Interpolation Capabilities for Function Approximations*, IEEE Trans. Neural Networks, Vol. 5, No. 6, 1994, pp. 936-944.

[41] T. CHOW, AND C. T. LEUNG, *Performance Enhancement Using Nonlinear Preprocessing*, IEEE Trans. Neural Networks, Vol. 7, No. 4, 1996, pp. 1039-1042.

[42] C. C. CHUANG, S. F. SU, AND C. C. HSIAO, *The Annealing Robust Backpropagation (ARBP) Learning Algorithm*, IEEE Trans. Neural Networks, Vol. 11, No. 5, 2000, pp. 1067-1077.

[43] A. CICHOKI, AND R. UNBEHAUEN, *Simplified Neural Networks for Solving Linear Least Squares and Total Least Squares Problems in Real Time*, IEEE Trans. Neural Networks, Vol. 5, No. 6, 1994, pp. 910-923.

[44] I. B. CIOCOIU, *Time Series Analysis Using RBF Networks with FIR/IIR Synapses*, Neurocomputing 20 (1998), pp. 57-66.

[45] I. B. CIOCOIU, *RBF Networks Training Using a Dual Extended Kalman Filter*, Neurocomputing 48 (2002), pp. 609-622.

[46] D. S. CLOUSE, C. L. GILES, B. G. HORNE, AND G. W. COTTRELL, *Time-Delay Neural Networks: Representation and Induction of Finite-State Machines*, IEEE Trans. Neural Networks, Vol. 8, No. 5, 1997, pp. 1065-1070.

[47] J. T. CONNOR, R. D. MARTIN, AND L. E. ATLAS, *Recurrent Neural Networks and Robust Time Series Prediction*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 240-253.

[48] A. J. CONWAY, K. P. MACPHERSON, J. C. BROWN, *Delayed Time Series Predictions with Neural Networks*, Neurocomputing 18 (1998), pp. 81-89.

[49] E. M. CORWIN, A. M. LOGAR, AND W. J. B. OLDHAM, *An Iterative Method for Training Multilayer Networks with Threshold Functions*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 1994, pp. 507-508.

[50] N. E. COTTER, *The Stone-Weierstrass Theorem and Its Application to Neural Networks*, IEEE Trans. Neural Networks, Vol. 1, No. 4, 1990, pp. 290-295.

[51] M. COTTRELL, B. GIRARD, Y. GIRARD, M. MANGEAS, AND C. MULLER, *Neural Modeling for Time Series: A Statistical Stepwise Mehtod for Weight Elimination*, IEEE Trans. Neural Networks, Vol. 6, No. 6, 1995, pp. 1355-1364.

[52] P. E. CROUCH, *Hamiltonian Realizations of Finite Volterra Series*, in: Theory and Applications of Control Systems, eds. C. I. Byrnes and A. Lindquist, North-Holland, 1986, pp. 247-259.

[53] G. CYBENKO, *Approximation by Superpositions of a Sigmoidal Function*, Mathematics of Control, Signals and Systems (1989) vol. 2 no. 4 : 303-314.

[54] R. DE LEONE, R. CAPPARUCCIA, AND E. MERELLI, *A Successive Overrelaxation Backpropagation Algorithm for Neural-Network Training*, IEEE Trans. Neural Networks, Vol. 9, No. 3, 1998, pp. 381-388.

[55] C. M. DENEGRI, JR., *Limit Cycle Oscillation Flight Test Results of a Fighter with External Stores*, Journal of Aircraft, Vol. 36, No. 5, 2000, pp. 761-769.

[56] C. M. DENEGRI, JR., AND M. A. CUTCHINS, *Evaluation of Classical Flutter Analyses for the Prediction of Limit Cycle Oscillations*, AIAA-97-1021.

[57] C. M. DENEGRI, JR., AND M. R. JOHNSON, *Limit Cycle Oscillation Prediction Using Artificial Neural Networks*, Journal of Guidance, Control, and Dynamics, Vol. 24, No. 5, 2001, pp. 887-895.

[58] M. DI MARTINO, S. FANELLI, AND M. PROTASI, *Exploring and Comparing the Best "Direct Methods" for the Efficient Training of MLP-Networks*, IEEE Trans. Neural Networks, Vol. 7, No. 6, 1996, pp. 1497-1502.

[59] A. DOERING, M. GALICKI, AND H. WITTE, *Structure Optimization of Neural Networks with the A^*-Algorithm*, IEEE Trans. Neural Networks, Vol. 8, No. 6, 1997, pp. 1434-1445.

[60] H. DRUCKER, AND Y. L. CUN, *Improving Generalization Performance Using Double Backpropagation*, IEEE Trans. Neural Networks, Vol. 3, No. 6, 1992, pp. 991-997.

[61] A. DUMITRAŞ, *Artificial Neural Network Design* (romanian language), Odeon Publishing House, Bucharest, Romania, 1997.

[62] R. J. DURO, AND J. S. REYES, *Discrete-Time Backpropagation for Training Synaptic Delay-Based Artificial Neural Networks*, IEEE Trans. Neural Networks, Vol. 10, No. 4, 1999, pp. 779-789.

[63] J. L. ELMAN, *Finding Structure in Time*, Cognitive Science, No. 14, 1990, pp. 179-211.

[64] A. P. ENGELBRECHT, *A New Pruning Heuristic Based on Variance Analysis of Sensitivity Information*, IEEE Trans. Neural Networks, Vol. 12, No. 6, 2001, pp. 1386-1399.

[65] R. ENNS, AND J. SI, *Apache Helicopter Stabilization Using Neural Dynamic Programming*, Journal of Guidance, Control, and Dynamics, Vol. 25, No. 1, 2002, pp. 19-25.

[66] S. ERGEZINGER, AND E. THOMSEN, *An Accelerated Learning Algorithm for Multilayer Perceptrons: Optimization Layer by Layer*, IEEE Trans. Neural Networks, Vol. 6, No. 1, 1995, pp. 31-42.

[67] S. Y. FAKHOURI, AND M. G. MYLROI, *Identification and Realisation of Multi-Dimensional Kernels*, in: Third IMA Conference in Control Theory, eds. J. E. Marshall, W. D. Collins, C. J. Harris, and D. H. Owens, Academic Press, 1981, pp. 65-86.

[68] J. FAN, AND Q. YAO, *Nonlinear Time Series - Nonparametric and Parametric Methods*, Springer-Verlag, New York, 2003.

[69] S. FERRARI, AND R. F. STENGEL, *Classical/Neural Synthesis of Nonlinear Control Systems*, AIAA-2000-4552, Proceedings of the 2000 AIAA Guidance, Navigation, and Control Conference, Denver, CO, Aug. 14-17, 2000.

[70] S. FERRARI, AND R. F. STENGEL, *Algebraic Training of a Neural Network*, Proceedings of the 2001 American Control Conference, Arlington, VA, June 2001.

[71] D. B. FOGEL, *An Information Criterion for Optimal Neural Network Selection*, IEEE Trans. Neural Networks, Vol. 2, No. 5, 1991, pp. 490-497.

[72] L. M. FU, H. H. HSU, AND J. C. PRINCIPE, *Incremental Backpropagation Learning Networks*, IEEE Trans. Neural Networks, Vol. 7, No. 3, 1996, pp. 757-761.

[73] Y. FUKUOKA, H. MATSUKI, H. MINAMITANI, A. ISHIDA, *A Modified Back-Propagation Method to Avoid False Local Minima*, Neural Networks, Vol. 11, 1998, pp. 1059-1072.

[74] G. E. FULCHER, AND D. E. BROWN, *A Polynomial Network for Predicting Temperature Distributions*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 1994, pp. 372-379.

[75] K. I. FUNAHASHI, *On the Approximate Realization of Continuous Mappings by Neural Networks*, Neural Networks, Vol. 2, No. 3, 1989, pp. 183-192.

[76] A. R. GALLANT, AND H. WHITE, *On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks*, Neural Networks, Vol. 5, 1992, pp. 129-138.

[77] A. R. GALLANT, AND H. WHITE, *There Exists a Neural Network that Does Not Make Avoidable Mistakes*, Proc. IEEE Int. Conf. Neural Networks, San Diego, CA, July 24-27, 1998, Vol. I, pp. 657-664.

[78] A. B. GEVA, *ScaleNet – Multiscale Neural Network Architecture for Time Series Prediction*, IEEE Trans. Neural Networks, Vol. 9, No. 5, 1998, pp. 1471-1482.

[79] C. L. GILES, D. CHEN, G. Z. SUN, H. H. CHEN, Y. C. LEE, AND M. W. GOUDREAU, *Constructive Learning of Recurrent Neural Networks: Limitations of Recurrent Cascade Correlation and A Simple Solution*, IEEE Trans. Neural Networks, Vol. 6, No. 4, 1995, pp. 829-835.

[80] D. E. GOLDBERG, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[81] J. B. GOMM, AND D. L. YU, *Selecting Radial Basis Function Network Centers with Recursive Orthogonal Least Square Training*, IEEE Trans. Neural Networks, Vol. 11, No. 2, 2000, pp. 306-314.

[82] H. GU, AND H. TAKAHASHI, *Towards More Practical Average Bounds on Supervised Learning*, IEEE Trans. Neural Networks, Vol. 7, No. 4, 1996, pp. 953-968.

[83] M. GÜLER, *A Model with an Intrinsic Property of Learning Higher Order Correlations*, Neural Networks, Vol. 14, 2001, pp. 495-504.

[84] M. GUO, Z. BAI AND H. Z. AN, *Multi-Step Prediction for Nonlinear Autoregressive Models Based on Empirical Distributions*, Statistica Sinica, No. 9, 1999, pp. 559-570.

[85] K. GURNEY, *Neural Nets*, www.shef.ac.uk/psychology/gurney/notes/, Online Course Notes.

[86] M. T. HAGAN, H. B. DEMUTH, AND M. BEALE, *Neural Network Design*, PWS Publishing Company, 1996.

[87] M. T. HAGAN, AND M. B. MENHAJ, *Training Feedforward Neural Networks with the Marquardt Algorithm*, IEEE Trans. Neural Networks, Vol. 5, No. 6, 1994, pp. 989-993.

[88] V. HAGGAN, AND T. OZAKI, *Amplitude-Dependent Exponential AR Model Fitting for Non-Linear Random Vibrations*, in: Time Series, ed. O. D. Anderson, North-Holland, 1980, pp. 57-71.

[89] V. HAGGAN, AND T. OZAKI, *Modelling Nonlinear Random Vibrations Using an Amplitude-Dependent Autoregressive Time Series Model*, Biometrika, Vol. 68, No. 1, 1981, pp. 189-196.

[90] J. V. Hansen, and R. D. Nelson, *Neural Networks and Traditional Time Series Methods: A Sinergistic Combination in State Economic Forecasts*, IEEE Trans. Neural Networks, Vol. 8, No. 4, 1997, pp. 863-873.

[91] S. Haykin, and L. Li, *Nonlinear Adaptive Prediction of Nonstationary Signals*, IEEE Trans. Signal Processing, Vol. 43, No. 2, 1995, pp. 526-535.

[92] D. P. Helmbold, J. Kivinen, and M. K. Warmuth, *Relative Loss Bounds for Single Neurons*, IEEE Trans. Neural Networks, Vol. 10, No. 6, 1999, pp. 1291-1304.

[93] K. Hirasawa, S. H. Kim, J. Hu, J. Murata, M. Han, and C. Jin, *Improvement of Generalization Ability for Identifying Dynamical Systems by Using Universal Learning Networks*, Neural Networks, Vol. 14, 2001, pp. 1389-1404.

[94] C. C. Holmes, and B. K. Mallick, *Bayesian Wavelet Networks for Nonparametric Regression*, IEEE Trans. Neural Networks, Vol. 11, No. 1, 2000, pp. 27-35.

[95] X. Hong, and S. A. Billings, *Time Series Multistep-Ahead Predictability Estimation and Ranking*, Journal of Forecasting, 18 (1999), pp. 139-149.

[96] K. Hornik, *Approximation Capabilities of Multilayer Feedforward Networks*, Neural Networks, Vol. 4, 1991, pp. 251-257.

[97] K. Hornik, M. Stinchcombe, and H. White, *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, Vol. 2, No. 5, 1989, pp. 359-366.

[98] S. Hosseini, and C. Jutten, *Maximum Likelihood Neural Approximation in Presence of Additive Colored Noise*, IEEE Trans. Neural Networks, Vol. 13, No. 1, 2002, pp. 117-131.

[99] T. C. R. Hsiao, C. W. Lin, H. K. Chiang, *Partial Least-Squares Algorithm for Weights Initialization of Backpropagation Network*, Neurocomputing, Vol. 50, 2003, pp. 237-247.

[100] N. E. Huang, Z. Shen, and S. R. Long, *A New View of Nonlinear Water Waves: The Hilbert Spectrum*, Annual Review of Fluid Mechanics, Vol. 31, 1999, pp. 417-457.

[101] N. E. Huang, Z. Shen, S. R. Long, M. C. Wu, H. H. Shih, Q. Zheng, N. C. Yen, C. C. Tung, and H. H. Liu, *The Empirical Mode Decomposition and the Hilbert Spectrum for Nonlinear and Non-Stationary Time Series Analysis*, R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci., Vol. 454, No. 1971, 1998, pp.903-995.

[102] M. Hüsken, and P. Stagge, *Recurrent Neural Networks for Time Series Classification*, Neurocomputing, No. 50, 2003, pp. 223-235.

[103] J. N. Hwang, S. R. Lay, M. Maechler, R. D. Martin, and J. Schmiert, *Regression Modeling in Back-Propagation and Projection Pursuit Learning*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 1994, pp. 342-353.

[104] M. M. ISLAM, AND K. MURASE, *A New Algorithm to Design Compact Two-Hidden-Layer Artificial Neural Networks*, Neural Networks, Vol. 14, 2001, pp. 1265-1278.

[105] Y. ITO, *Representation of Functions by Superposition of a Step or Sigmoid Function and Their Applications to Neural Network Theory*, Neural Networks, Vol. 4, 1991, pp. 385-394.

[106] J. S. N. JEAN, AND J. WANG, *Weight Smoothing to Improve Network Generalization*, IEEE Trans. Neural Networks, Vol. 5, No. 5, 1994, pp. 752-763.

[107] D. JIANG, AND J. WANG, *On-Line Learning of Dynamical Systems in the Presence of Model Mismatch and Disturbances*, IEEE Trans. Neural Networks, Vol. 11, No. 6, 2000, pp. 178-183.

[108] K. C. JIM, C. L. GILES, AND B. G. HORNE, *An Analysis of Noise in Recurrent Neural Networks: Convergence and Generalization*, IEEE Trans. Neural Networks, Vol. 7, No. 6, 1996, pp. 1424-1438.

[109] L. JIN, AND M. M. GUPTA, *Stable Dynamic Backpropagation Learning in Recurrent Neural Networks*, IEEE Trans. Neural Networks, Vol. 10, No. 6, 1999, pp. 1321-1334.

[110] J. D. JOHNSON, J. LU, A. P. DHAWAN, AND R. LIND, *Real-Time Identification of Flutter Boundaries Using the Discrete Wavelet Transform*, Journal of Guidance, Control, and Dynamics, Vol. 25, No. 2, 2002, pp. 334-339.

[111] M. D. JOHNSON, AND K. ROKHSAZ, *Using Artificial Neural Networks and Self-Organizing Maps for Detection of Airframe Icing*, Journal of Aircraft, Vol. 38, No. 2, 2001, pp. 224-230.

[112] J. C. JUANG, *Stability Analysis of Hopfield-Type Neural Networks*, IEEE Trans. Neural Networks, Vol. 10, No. 6, 1999, pp. 1366-1374.

[113] P. V. KABAILA, *Estimation Based on One Step Ahead Prediction Versus Estimation Based on Multi-Step Ahead Prediction*, Stochastics, Vol. 6, 1981, pp. 43-55.

[114] S. V. KAMARTHI, AND S. PITTNER, *Accelerating Neural Network Training Using Weight Extrapolations*, Neural Networks, Vol. 12, 1999, pp. 1285-1299.

[115] P. P. KANJILAL, AND D. N. BANERJEE, *On the Application of Orthogonal Transformation for the Design and Analysis of Feedforward Networks*, IEEE Trans. Neural Networks, Vol. 6, No. 5, 1995, pp. 1061-1070.

[116] D. A. KARRAS, AND S. J. PERANTONIS, *An Efficient Constrained Training Algorithm for Feedforward Networks*, IEEE Trans. Neural Networks, Vol. 6, No. 6, 1995, pp. 1420-1433.

[117] G. N. KARYSTINOS, AND D. A. PADOS, *On Overfitting, Generalization, and Randomly Expanded Training Sets*, IEEE Trans. Neural Networks, Vol. 11, No. 5, 2000, pp. 1050-1057.

[118] P. Kerlirzin, and P. Réfrégier, *Theoretical Investigation of the Robustness of Multilayer Perceptrons: Analysis of the Linear Case and Extension to Nonlinear Networks*, IEEE Trans. Neural Networks, Vol. 6, No. 3, 1995, pp. 560-571.

[119] B. Kermanshahi, *Recurrent Neural Network for Forecasting Next 10 Years Loads of Nine Japanese Utilities*, Neurocomputing, Vol. 23 (1998), pp. 125-133.

[120] C. Khunasaraphan, K. Vanapipat, and C. Lursinsap, *Weight Shifting Techniques for Self-Recovery Neural Networks*, IEEE Trans. Neural Networks, Vol. 5, No. 4, 1994, pp. 651-658.

[121] D. Kim, and M. Marciniak, *Prediction of Vertical Tail Maneuver Loads Using Backpropagation Neural Networks*, Journal of Aircraft, Vol. 37, No. 3, 2000, pp. 526-530.

[122] S. S. Kim, *Time Delay Recurrent Neural Network for Temporal Correlations and Predictions*, Neurocomputing, Vol. 20 (1998), pp. 253-263.

[123] H. R. Kirby, S. M. Watson, M. S. Dougherty, *Should we use neural networks or statistical models for short-term motorway traffic forecasting ?*, International Journal of Forecasting 13 (1997) : 43-50.

[124] J. Ko, A. J. Kurdila, and T. W. Strganac, *Nonlinear Control of a Prototypical Wing Section with Torsional Nonlinearity*, Journal of Guidance, Control, and Dynamics, Vol. 20, No. 6, 1997, pp. 1181-1189.

[125] J. Ko, T. W. Strganac, and A. J. Kurdila, *Stability and Control of a Structurally Nonlinear Aeroelastic System*, Journal of Guidance, Control, and Dynamics, Vol. 21, No. 5, 1998, pp. 718-725.

[126] E. B. Kosmatopoulos, M. M. Polycarpou, M. A. Christodoulou, and P. A. Ioannou, *High-Order Neural Network Structures for Identification of Dynamical Systems*, IEEE Trans. Neural Networks, Vol. 6, No. 2, 1995, pp. 422-431.

[127] A. Kowalczyk, and H. L. Ferrá, *Developing Higher-Order Networks with Empirically Selected Units*, IEEE Trans. Neural Networks, Vol. 5, No. 5, 1994, pp. 698-711.

[128] S. C. Kremer, *On the Computational Power of Elman-Style Recurrent Networks*, IEEE Trans. Neural Networks, Vol. 6, No. 4, 1995, pp. 1000-1004.

[129] C. C. Ku, and K. Y. Lee, *Diagonal Recurrent Neural Networks for Dynamic Systems Control*, IEEE Trans. Neural Networks, Vol. 6, No. 1, 1995, pp. 144-156.

[130] K. W. C. Ku, M. W. Mak, and W. C. Siu, *Adding Learning to Cellular Genetic Algorithms for Training Recurrent Neural Networks*, IEEE Trans. Neural Networks, Vol. 10, No. 2, 1999, pp. 239-252.

[131] C. M. KUAN, *A Recurrent Newton Algorithm and Its Convergence Properties*, IEEE Trans. Neural Networks, Vol. 6, No. 3, 1995, pp. 779-783.

[132] A. J. KURDILA, R. J. PRAZENICA, O. REDINIOTIS, AND T. W. STRGANAC, *Multiresolution Methods for Reduced-Order Models for Dynamical Systems*, Journal of Guidance, Control, and Dynamics, Vol. 24, No. 2, 2001, pp. 193-200.

[133] A. J. KURDILA, T. W. STRGANAC, J. L. JUNKINS, J. KO, AND M. R. AKELLA, *Nonlinear Control Methods for High-Energy Limit-Cycle Oscillations*, Journal of Guidance, Control, and Dynamics, Vol. 24, No. 1, 2001, pp. 185-192.

[134] T. M. KWON, AND H. CHENG, *Contrast Enhancement for Backpropagation*, IEEE Trans. Neural Networks, Vol. 7, No. 2, 1996, pp. 515-523.

[135] T. M. KWON, AND E. H. FEROZ, *A Multilayered Perceptron Approach to Prediction of the SEC's Investigation Targets*, IEEE Trans. Neural Networks, Vol. 7, No. 5, 1996, pp. 1286-1290.

[136] M. M. LAMEGO, *Adaptive Structures with Algebraic Loops*, IEEE Trans. Neural Networks, Vol. 12, No. 1, 2001, pp. 33-42.

[137] K. J. LANG, G. E. HINTON, *A Time-Delay Neural Network Architecture for Speech Recognition*, CMU-CS-88-152, 1998.

[138] B. LEBARON, AND A. S. WEIGEND, *A Bootstrap Evaluation of the Effect of Data Splitting on Financial Time Series*, IEEE Trans. Neural Networks, Vol. 9, No. 1, 1998, pp. 213-219.

[139] T. LEE, AND Y. KIM, *Nonlinear Adaptive Flight Control Using Backstepping and Neural Networks Controller*, Journal of Guidance, Control, and Dynamics, Vol. 24, No. 4, 2001, pp. 675-682.

[140] B. H. K. LEE, S. J. PRICE, AND Y. S. WONG, *Nonlinear Aeroelastic Analysis of Airfoils: Bifurcation and Chaos*, Progress in Aerospace Sciences, Vol. 35, No. 3, 1999, pp. 205-334.

[141] C. S. LEUNG, A. C. TSOI, AND L. W. CHAN, *Two Regularizers for Recursive Least Squared Algorithms in Feedforward Multilayered Neural Networks*, IEEE Trans. Neural Networks, Vol. 12, No. 6, 2001, pp. 1314-1332.

[142] C. S. LEUNG, G. H. YOUNG, J. SUM, AND W. K. KAN, *On the Regularization of Forgetting Recursive Least Square*, IEEE Trans. Neural Networks, Vol. 10, No. 6, 1999, pp. 1482-1486.

[143] A. U. LEVIN, AND K. S. NARENDRA, *Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilization*, IEEE Trans. Neural Networks, Vol. 4, No. 2, 1993, pp. 192-206.

C

[144] A. U. Levin, and K. S. Narendra, *Control of Nonlinear Dynamical Systems Using Neural Networks - Part II: Observability, Identification, and Control*, IEEE Trans. Neural Networks, Vol. 7, No. 1, 1996, pp. 30-42.

[145] E. Levin, *Hidden Control Neural Architecture Modeling of Nonlinear Time Varying Systems and Applications*, IEEE Trans. Neural Networks, Vol. 4, No. 1, 1993, pp. 109-116.

[146] Y. Li, N. Sundararajan, and P. Saratchandran, *Stable Neuro-Flight-Controller Using Fully Tuned Radial Basis Function Neural Networks*, Journal of Guidance, Control, and Dynamics, Vol. 24, No. 4, 2001, pp. 665-674.

[147] T. Lin, B. G. Horne, and C. L. Giles, *How Embedded Memory in Recurrent Neural Network Architectures Helps Learning Long-Term Temporal Dependencies* Neural Networks, Vol. 11, 1998, pp. 861-868.

[148] T. Lin, B. G. Horne, Peter Tiňo, and C. L. Giles, *Learning Long-Term Dependencies in NARX Recurrent Neural Networks*, IEEE Trans. Neural Networks, Vol. 7, No. 6, 1996, pp. 1329-1338.

[149] D. J. Linse, and R. F. Stengel, *Identification of Aerodynamic Coefficients Using Computational Neural Networks*, Journal of Guidance, Control, and Dynamics, Vol. 16, No. 6, 1993.

[150] G. P. Liu, V. Kadirkamanathan, S. A. Billings, *On-Line Identification of Nonlinear Systems using Volterra Polynomial Basis Function Neural Networks*, Neural Networks, Vol. 11, 1998, pp. 1645-1657.

[151] L. Liu, *Mathematical Analysis in Nonlinear Aeroelasticity*, Ph.D. Dissertation, Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta, Canada, December 2001.

[152] J. T. Lo, and D. Bassu, *Adaptive Multilayer Perceptrons With Long- and Short-Term Memories*, IEEE Trans. Neural Networks, Vol. 13, No. 1, 2002, pp. 22-33.

[153] J. T. H. Lo, *Synthetic Approach to Optimal Filtering*, IEEE Trans. Neural Networks, Vol. 5, No. 5, 1994, pp. 803-811.

[154] B. Lukovic, P. Orkwis, M. Turner, and B. Sekar, *Effect of Cavity L/D Variations on Neural Network-Based Deterministic Unsteadiness Source Terms*, AIAA Paper 2002-0857, Proceedings of the 40th Aerospace Sciences Meeting and Exhibit, Reno, Nevada, Jan 14-17, 2002.

[155] S. Ma, and C. Ji, *Fast Training of Recurrent Networks Based on the EM Algorithm*, IEEE Trans. Neural Networks, Vol. 9, No. 1, 1998, pp. 11-26.

[156] O. MAAS, J. P. BOULANGER, AND S. THIRIA, *Use of Neural Networks for Predictions Using Time Series: Illustration with the El Niño Southern Oscillation Phenomenon*, Neurocomputing, Vol. 30, 2000, pp. 53-58.

[157] G. D. MAGOULAS, V. P. PLAGIANAKOS, AND M. N. VRAHATIS, *Globally Convergent Algorithms With Local Learning Rates*, IEEE Trans. Neural Networks, Vol. 13, No. 3, 2002, pp. 774-779.

[158] D. P. MANDIC, AND J. A. CHAMBERS, *Toward an Optimal PRNN-based Nonlinear Predictor*, IEEE Trans. Neural Networks, Vol. 10, No. 6, 1999, pp. 1435-1441.

[159] D. P. MANDIC, AND J. A. CHAMBERS, *Exploiting Inherent Relationships in RNN Architectures*, Neural Networks, Vol. 12, 1999, pp. 1341-1345.

[160] D. P. MANDIC, AND J. A. CHAMBERS, *On the Choice of Parameters of the Cost Function in Nested Modular RNN's*, IEEE Trans. Neural Networks, Vol. 11, No. 2, 2000, pp. 315-321.

[161] N. MARCO, S. LANTERI, *A Two-Level Parallelization Strategy for Genetic Algorithms Applied to Optimum Shape Design*, Parallel Computing, 26 (2000), pp. 377-397.

[162] J. L. MARROQUIN, *Measure Fields for Function Approximation*, IEEE Trans. Neural Networks, Vol. 6, No. 5, 1995, pp. 1081-1090.

[163] T. MASTERS, *Practical Neural Network Recipes in C++*, Academic Press, Boston, 1993.

[164] W. S. MCCULLOCH, AND W. PITTS, *A Logical Calculus of the Ideas Immanent in Nervous Activity*, Bulletin of Mathematical Biophysics, Vol. 5, 1943, pp. 115-133.

[165] M. B. MCFARLAND, AND A. J. CALISE, *Neural Networks and Adaptive Nonlinear Control of Agile Antiair Missiles*, Journal of Guidance, Control, and Dynamics, Vol. 23, No. 3, 2000, pp. 547-553.

[166] J. MCNAMES, *Local Averaging Optimization for Chaotic Time Series Prediction*, Neurocomputing, Vol. 48, 2002, pp. 279-297.

[167] M. C. MEDEIROS, AND A. VEIGA, *A Hybrid Linear-Neural Model for Time-Series Forecasting*, IEEE Trans. Neural Networks, Vol. 11, No. 6, 2000, pp. 1402-1412.

[168] M. MIGNOTTE, C. COLLET, P. PÉREZ, AND P. BOUTHÉMY, *Hybrid Genetic Optimization and Statistical Model-Based Approach for the Classification of Shadow Shapes in Sonar Imagery*, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 22, No. 2, 2000, pp. 129-141.

[169] R. R. MOHLER, *Nonlinear Systems, Vol. II, Applications to Bilinear Control*, Prentice Hall, 1991, pp. 93-113.

[170] N. Murata, S. Yoshizawa, and S. I. Amari, *Network Information Criterion - Determining the Number of Hidden Units for an Artificial Neural Network Model*, IEEE Trans. Neural Networks, Vol. 5, No. 6, 1994, pp. 865-871.

[171] A. F. Murray and P. J. Edwards, *Enhanced MLP Performance and Fault Tolerance Resulting from Synaptic Weight Noise During Training*, IEEE Trans. Neural Networks, Vol. 5, No. 5, 1994, pp. 792-802.

[172] K. S. Narendra, and S. Mukhopadhyay, *Adaptive Control Using Neural Networks and Approximate Models*, IEEE Trans. Neural Networks, Vol. 8 No. 3, 1997, pp. 475-485.

[173] K. S. Narendra, and K. Parthasarathy, *Identification and Control of Dynamical Systems Using Neural Networks*, IEEE Trans. Neural Networks, Vol. 1, No. 1, 1990, pp. 4-26.

[174] K. S. Narendra, and K. Parthasarathy, *Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks*, IEEE Trans. Neural Networks, Vol. 2, No. 2, 1991, pp. 252-262.

[175] O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz, and G. Dreyfus, *Training Recurrent Neural Networks: Why and How? An Illustration in Dynamical Process Modeling*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 178-183.

[176] A. Niederlinski, *Error Convergence Rate for Multiple-Recursion RLS Estimation in Linear Multi-Step Prediction Models*, Archives of Control Sciences, Vol. 4 (XL), No. 3-4, 1995, pp.173-201.

[177] N. Y. Nikolaev, and Hitoshi Iba, *Learning Polynomial Feedforward Neural Networks by Genetic Programming and Backpropagation*, IEEE Trans. Neural Networks, Vol. 14, No. 2, 2003, pp. 337-350.

[178] J. Nishii, *A Learning Model for Oscillatory Networks*, Neural Networks, Vol. 11, 1998, pp. 249-257.

[179] M. Nørgaard, *Neural Networks for Modelling and Control of Dynamic Systems : A Practitioner's Handbook*, Springer, Berlin, New York, 2000.

[180] S. H. Oh, *Improving the Error Backpropagation Algorithm with a Modified Error Function*, IEEE Trans. Neural Networks, Vol. 8, No. 3, 1997, pp. 799-802.

[181] O. Olurotimi, *Recurrent Neural Network Training with Feedforward Complexity*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 185-197.

[182] T. Ozaki, *Non-Linear Time Series Models for Non-Linear Random Vibrations*, Journal of Applied Probability, Vol. 17, 1980, pp. 84-93.

[183] T. Ozaki, *The Statistical Analysis of Perturbed Limit Cycle Processes Using Non-Linear Time Series Models*, Journal of Time Series Analysis, Vol. 3, No. 1, 1982, pp. 29-40.

[184] R. Parisi, E. D. Di Claudio, G. Orlandi, and B. D. Rao, *A Generalized Learning Paradigm Exploiting the Structure of Feedforward Neural Networks*, IEEE Trans. Neural Networks, Vol. 7, No. 6, 1996, pp. 1450-1460.

[185] T. Parisini, and R. Zoppoli, *Neural Networks for Feedback Feedforward Nonlinear Control Systems*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 1994, pp. 436-449.

[186] A. G. Parlos, K. T. Chong, and A. F. Atiya, *Application of the Recurrent Multilayer Perceptron in Modeling Complex Process Dynamics*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 255-265.

[187] A. G. Parlos, B. Fernandez, A. F. Atiya, J. Muthusami, and W. K. Tsai, *An Accelerated Learning Algorithm for Multilayer Perceptron Networks*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 1994, pp. 493-497.

[188] A. G. Parlos, O. T. Rais, and A. F. Atiya, *Multi-Step-Ahead Prediction Using Dynamic Recurrent Neural Networks*, Neural Networks 13 (2000), pp. 765-786.

[189] S. J. Perantonis, and P. J. G. Lisboa, *Translation, Rotation, and Scale-Invariant Pattern Recognition by High-Order Neural Networks and Moment Classifiers*, IEEE Trans. Neural Networks, Vol. 3, No. 2, 1992, pp. 241-251.

[190] V. V. Phansalkar, and P. S. Sastry, *Analysis of the Back-Propagation Algorithm with Momentum*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 199[illegible]p. 505-506.

[191] D. S. Phatak, and I. Koren, *Connectivity and Performance Tradeoffs in the Cascade Correlation Learning Architecture*, IEEE Trans. Neural Networks, Vol. 5, No. 6, 1994, pp. 930-935.

[192] S. W. Piché, *Steepest Descent Algorithms for Neural Network Controllers and Filters*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 198-212.

[193] E. Polak, *Opti[illegible]ation - Algorithms and Consistent Approximations*, Springer-Verlag, New York 1997.

[194] C. Popescu, and Y. S. Wong, *A Nonlinear Statistical Approach for Aeroelastic Response Prediction*, AIAA paper 2002-1281, Proceedings of the 43rd AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference, Denver, Colorado, Apr 22-25, 2002.

[195] G. V. Puskorius, and L. A. Feldkamp, *Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 279-297.

[196] S. Z. Qin, H. T. Su, and T. J. McAvoy, *Comparison of Four Neural Net Learning Methods for Dynamic System Identification*, IEEE Trans. Neural Networks, Vol. 3, No. 1, 1992, pp. 122-130.

[197] M. M. Rai, and N. K. Madavan, *Aerodynamic Design Using Neural Networks*, AIAA Journal, Vol. 38, No. 1, 2000, pp. 173-182.

[198] V. Ramamurti, and J. Ghosh, *Structurally Adaptive Modular Networks for Nonstationary Environments*, IEEE Trans. Neural Networks, Vol. 10, No. 1, 1999, pp. 152-159.

[199] R. Reed, R. J. Marks II, and S. Oh, *Similarities of Error Regularization, Sigmoid Gain Scaling, Target Smoothing, and Training with Jitter*, IEEE Trans. Neural Networks, Vol. 6, No. 3, 1995, pp. 152-159.

[200] K. Rohani, M. S. Chen, and M. T. Manry, *Neural Subnet Design by Direct Polynomial Mapping*, IEEE Trans. Neural Networks, Vol. 3, No. 6, 1992, pp. 1024-1026.

[201] T. J. Rohloff, and I. Catton, *Fault Tolerance and Extrapolation Stability of a Neural Network Air-Data Estimator*, Journal of Aircraft, Vol. 36, No. 3, 1999, pp. 571-576.

[202] I. Rojas, H. Pomares, J. L. Bernier, J. Ortega, B. Pino, F. J. Pelayo, A. Prieto, *Time Series Analysis Using Normalized PG-RBF Network With Regression Weights*, Neurocomputing, Vol. 42, 2002, pp. 267-285.

[203] Z. Roth, and Y. Baram, *Multidimensional Density Shaping by Sigmoids*, IEEE Trans. Neural Networks, Vol. 7, No. 5, 1996, pp. 1291-1298.

[204] C. W. Rowley, T. Colonius, and R. M. Murray, *Dynamical Models for Control of Cavity Oscillations*, AIAA Paper 2001-2126.

[205] P. RoyChowdhury, Y. P. Singh, and R. A. Chansarkar, *Dynamic Tunnelling Technique for Efficient Training of Multilayer Perceptrons*, IEEE Trans. Neural Networks, Vol. 10, No. 1, 1999, pp. 48-55.

[206] A. Ruiz, D. H. Owens, and S. Townley, *Existence, Learning, and Replication of Periodic Motions in Recurrent Neural Networks*, IEEE Trans. Neural Networks, Vol. 9, No. 4, 1998, pp. 651-661.

[207] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch, II, *Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks*, IEEE Trans. Neural Networks, Vol. 9, No. 6, 1998, pp. 1456-1470.

[208] N. SADEGH, *A Perceptron Network for Functional Identification and Control of Nonlinear Systems*, IEEE Trans. Neural Networks, Vol. 4, No. 6, 1993, pp. 982-988.

[209] M. SALMERÓN, J. ORTEGA, C. G. PUNTONET, AND A. PRIETO, *Improved RAN Sequential Prediction Using Orthogonal Techniques*, Neurocomputing, 41, 2001, pp. 153-172.

[210] A. SARANLI, AND B. BAYKAL, *Complexity Reduction in Radial Basis Function (RBF) Networks by Using Radial B-Spline Functions*, Neurocomputing 18 (1998), pp. 183-194.

[211] P. S. SASTRY, G. SANTHARAM, AND K. P. UNNIKRISHNAN, *Memory Neuron Networks for Identification and Control of Dynamical Systems*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 306-319.

[212] R. J. SCHILLING, J. J. CARROLL, AND A. F. AL-AJLOUNI, *Approximation of Nonlinear Systems with Radial Basis Function Neural Networks*, IEEE Trans. Neural Networks, Vol. 12, No. 1, 2001, pp. 1-15.

[213] R. C. SCOTT, AND L. E. PADO, *Active Control of Wind-Tunnel Model Aeroelastic Response Using Neural Networks*, Journal of Guidance, Control, and Dynamics, Vol. 23, No. 6, 2000, pp. 1100-1108.

[214] R. SETIONO, AND L. C. K. HUI, *Use of a Quasi-Newton Method in a Feedforward Neural Network Construction Algorithm*, IEEE Trans. Neural Networks, Vol. 6, No. 1, 1995, pp. 273-277.

[215] R. H. SHUMWAY, AND D. S. STOFFER, *Time Series Analysis and Its Applications*, Springer, New York, 2000.

[216] J. SI, AND A. N. MICHEL, *Analysis and Synthesis of a Class of Discrete-Time Neural Networks with Multilevel Threshold Neurons*, IEEE Trans. Neural Networks, Vol. 6, No. 1, 1995, pp. 105-116.

[217] D. SIMON, *Training Radial Basis Neural Networks with the Extended Kalman Filter*, Neurocomputing, Vol. 48, 2002, pp. 455-475.

[218] S. SOLTANI, *On the Use of the Wavelet Decomposition for Time Series Prediction*, Neurocomputing, 48, 2002, pp. 267-277.

[219] D. F. SPECHT, *A General Regression Neural Network*, IEEE Trans. Neural Networks, Vol. 2, No. 6, 1991, pp. 568-576.

[220] B. SRINIVASAN, U. R. PRASAD, AND N. J. RAO, *Back Propagation Through Adjoints for the Identification of Nonlinear Dynamic Systems Using Recurrent Neural Models*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 213-227.

[221] D. Srinivasan, *Evolving Artificial Neural Networks for Short-Term Load Forecasting*, Neurocomputing, Vol. 23, 1998, pp. 265-276.

[222] T. W. Strganac, J. Ko, D. E. Thompson, and A. J. Kurdila, *Identification and Control of Limit Cycle Oscillations in Aeroelastic Systems*, Journal of Guidance, Control, and Dynamics, Vol. 23, No. 6, 2000, pp. 1127-1133.

[223] J. Sum, C. S. Leung, G. H. Young, and W. K. Kan, *On the Kalman Filtering Method in Neural Network Training and Pruning*, IEEE Trans. Neural Networks, Vol. 10, No. 1, 1999, pp. 161-166.

[224] F. Takens, *Detecting Strange Attractors in Turbulence*, in: Dynamical Systems and Turbulence - Warwick 1980, Lecture Notes in Mathematics 898, Springer-Verlag, 1981, pp. 366-381.

[225] Y. Tan, [illegible]. Wang, and J. M. Zurada, *Nonlinear Blind Source Separation Using a Radial Basis Function Network*, IEEE Trans. Neural Networks, Vol. 12, No. 1, 2001, pp. 124-134.

[226] K. Tanaka, *An Approach to Stability Criteria of Neural-Network Control Systems*, IEEE Trans. Neural Networks, Vol. 7, No. 3, 1996, pp. 629-642.

[227] C. C. Teng, and B. W. Wah, *Automated Learning for Reducing the Configuration of a Feedforward Neural Network*, IEEE Trans. Neural Networks, Vol. 7, No. 5, 1996, pp. 1072-1085.

[228] G. C. Tiao and D. Xu, *Robustness of Maximum Likelihood Estimates for Multi-Step Predictions: The Exponential Smoothing Case*, Biometrika, Vol. 80, No. 3, 1993, pp. 623-641.

[229] M. Torii, and M. T. Hagan, *Stability of Steepest Descent with Momentum for Quadratic Functions*, IEEE Trans. Neural Networks, Vol. 13, No. 3, 2002, pp. 752-756.

[230] E. Trentin, *Networks with Trainable Amplitude of Activation Functions*, Neural Networks, Vol. 14, 2001, pp. 471-493.

[231] C. Tsitouras, *Neural Networks with Multidimensional Transfer Functions*, IEEE Trans. Neural Networks, Vol. 13, No. 1, 2002, pp. 222-228.

[232] A. C. Tsoi, and A. D. Back, *Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures*, IEEE Trans. Neural Networks, Vol. 5, No. 2, 1994, pp. 229-239.

[233] M. Tsujitani, and T. Koshimizu, *Neural Discriminant Analysis*, IEEE Trans. Neural Networks, Vol. 11, No. 6, 2000, pp. 1394-1400.

[234] C. TURCHETTI, M. CONTI, P. CRIPPA, AND S. ORCIONI, *On The Approximation of Stochastic Processes by Approximate Identity Neural Networks*, IEEE Trans. Neural Networks, Vol. 9, No. 6, 1998, pp. 1069-1084.

[235] N. UEDA, AND R. NAKANO, *Deterministic Annealing EM Algorithm*, Neural Networks, Vol. 11, 1998, pp. 271-282.

[236] L. VECCI, F. PIAZZA, AND A. UNCINI, *Learning and Approximation Capabilities of Adaptive Spline Activation Function Neural Networks*, Neural Networks, Vol. 11 (1998), pp. 259-270.

[237] B. VERMA, *Fast Training of Multilayer Perceptrons*, IEEE Trans. Neural Networks, Vol. 8, No. 6, 1997, pp. 1314-1320.

[238] O. VOITCU, AND Y. S. WONG, *A Neural Network Approach for Nonlinear Aeroelastic Analysis*, AIAA paper 2002-1286, Proceedings of the 43rd AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference, Denver, Colorado, Apr 22-25, 2002.

[239] O. VOITCU, AND Y. S. WONG, *Neural Network Approach for Nonlinear Aeroelastic Analysis*, Journal of Guidance, Control and Dynamics, Vol. 26, No. 1, 2003, pp. 99-106.

[240] O. VOITCU, AND Y. S. WONG, *An Improved Neural Network Model for Nonlinear Aeroelastic Analysis*, AIAA paper 2003-1493, Proceedings of the 44th AIAA/ ASME/ ASCE/ AHS Structures, Structural Dynamics, and Materials Conference, Norfolk, Virginia, Apr 7-10, 2003.

[241] O. VOITCU, AND Y. S. WONG, *Robust Feature Extraction Maximization by Neural Networks for Long-Term Prediction*, submitted to: Neurocomputing, 2004.

[242] O. VOITCU, AND Y. S. WONG, *On the Construction of a Nonlinear Recursive Predictor*, submitted to: Journal of Computational and Applied Mathematics, 2004.

[243] A. WAIBEL, T. HANAZAWA, G. HINTON, K. SHIKANO, AND K. J. LANG, *Phoneme Recognition Using Time-Delay Neural Networks*, IEEE Trans. Acoustic, Speech and Signal Processing, Vol. 37, No. 3, 1989, pp. 328-339.

[244] H. WAKUYA, AND J. M. ZURADA, *Bi-Directional Computing Architecture for Time Series Prediction*, Neural Networks, Vol. 14, 2001, pp. 1307-1321.

[245] E. A. WAN, *Temporal Backpropagation: An Efficient Algorithm for Finite Impulse Response Neural Networks*.

[246] E. A. WAN, *Time Series Prediction by Using a Connectionist Network with Internal Delay Lines*, in: Time Series Prediction: Forecasting the Future and Understanding the Past, eds. A. S. Weigend and N. A. Gershfeld, Santa Fe Institute for Studies in the Sciences of Complexity, Proc. Vol. XV, Addison-Wesley, 1993.

[247] C. WANG, AND J. PRINCIPE, *Training Neural Networks with Additive Noise in the Desired Signal*, IEEE Trans. Neural Networks, Vol. 10, No. 6, 1999, pp. 1511-1516.

[248] D. L. WANG, AND B. YUWONO, *Incremental Learning of Complex Temporal Patterns*, IEEE Trans. Neural Networks, Vol. 7, No. 6, 1996, pp. 1465-1481.

[249] G. J. WANG, AND C. C. CHEN, *A Fast Multilayer Neural-Network Training Algorithm Based on the Layer-By-Layer Optimizing Procedures*, IEEE Trans. Neural Networks, Vol. 7, No. 3, 1996, pp. 768-775.

[250] L. WANG, *Discrete-Time Convergence Theory and Updating Rules for Neural Networks with Energy Functions*, IEEE Trans. Neural Networks, Vol. 8, No. 2, 1997, pp. 445-447.

[251] Y. J. WANG, AND C. T. LIN, *Runge-Kutta Neural Network for Identification of Dynamical Systems in High Accuracy*, IEEE Trans. Neural Networks, Vol. 9, No. 2, 1998, pp. 294-307.

[252] S. WEAVER, L. BAIRD, AND M. POLYCARPOU, *Using Localizing Learning to Improve Supervised Learning Algorithms*, IEEE Trans. Neural Networks, Vol. 12, No. 5, 2001, pp. 1037-1045.

[253] A. R. WEBB, *Functional Approximation by Feed-Forward Networks: A Least-Squares Approach to Generalization*, IEEE Trans. Neural Networks, Vol. 5, No. 3, 1994, pp. 363-371.

[254] D. M. WEBER, AND D. P. CASASENT, *The Extended Piecewise Quadratic Neural Network*, Neural Networks, Vol. 11, 1998, pp. 837-850.

[255] A. S. WEIGEND, B. A. HUBERMAN, D. E. RUMELHART, *Predicting the Future: A Connectionist Approach*, International Journal of Neural Systems, Vol. 1, No. 3 (1990), pp.193-209.

[256] N. WEYMAERE, AND J. P. MARTENS, *On the Initialization and Optimization of Multilayer Perceptrons*, IEEE Trans. Neural Networks, Vol. 5, No. 5, 1994, pp. 738-751.

[257] Y. S. WONG, B. H. K. LEE, AND T. K. S. WONG, *Parameter Extraction by Parallel Neural Networks*, Intelligent Data Analysis, No. 5, 2001, pp.59-71.

[258] Y. S. WONG, C. POPESCU, AND O. VOITCU, *Nonlinear Dynamic Prediction and Feature Extraction*, Proceedings of the 48th Annual Conference of Canadian Aeronautics and Space Institute, Toronto, 2001, pp. 97-106.

[259] Y. S. WONG, O. VOITCU, AND C. POPESCU, *An Expert Data Mining System for Flutter Boundary Prediction*, Proceedings of the 23th Congress of the International Council for Aeronautical Sciences (ICAS), Toronto, Sep 8-13, 2002.

[260] Y. S. WONG, O. VOITCU, AND C. POPESCU, *Data Mining Approach for Nonlinear Dynamic Predictions*, Proceedings of the Third International Workshop on Scientific Computing and Applications, City University of Hong Kong, Jan 6-9, 2003.

[261] J. Y. F. YAM, AND T. W. S. CHOW, *Extended Least Squares Based Algorithm for Training Feedforward Networks*, IEEE Trans. Neural Networks, Vol. 8, No. 3, 1997, pp. 806-810.

[262] Y. YAMAMOTO, AND P. N. NIKIFORUK, *A New Supervised Learning Algorithm for Multilayered and Interconnected Neural Networks*, IEEE Trans. Neural Networks, Vol. 11, No. 1, 2000, pp. 36-46.

[263] S. M. YANG, AND G. S. LEE, *Structural Damage Identification Using Pole/Zero Dynamics in Neural Networks*, AIAA Journal, Vol. 39, No. 9, 2001, pp. 1805-1807.

[264] J. YAO, AND C. L. TAN, *A Case Study on Using Neural Networks to Perform Technical Forecasting of Forex*, Neurocomputing 34 (2000), pp. 79-98.

[265] X. H. YU, G. A. CHEN, AND S. X. CHENG, *Dynamic Learning Rate Optimization of the Backpropagation Algorithm*, IEEE Trans. Neural Networks, Vol. 6, No. 3, 1995, pp. 669-677.

[266] J. M. ZAMARREÑO, AND P. VEGA, *State Space Neural Network: Properties and Application*, Neural Networks, Vol. 11, 1998, pp. 1099-1112.

[267] J. ZARZYCKI, *Orthogonal Ladder-Form Representations of Nonlinear Prediction Filters of the Volterra-Wiener Class*, Theory and Applications of Nonlinear Control Systems, C. I. Byrnes and A. Lindquist (editors), Elsevier Science Publishers B. V. (North-Holland), 1986, pp.421-435.

[268] P. ZEGERS, AND M. K. SUNDARESHAN, *Trajectory Generation and Modulation Using Dynamic Neural Networks*, IEEE Trans. Neural Networks, Vol. 14, No. 3, 2003, pp. 520-533.

[269] G. P. ZHANG, *Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model*, Neurocomputing 50 (2003), pp. 159-175.

[270] M. ZHANG, S. XU, AND J. FULCHER, *Neuron-Adaptive Higher-Order Neural Network Models for Automated Financial Data Modeling*, IEEE Trans. Neural Networks, Vol. 13, No. 1, 2002, pp. 188-204.

[271] S. ZHANG, X. ZHU, AND L. H. ZOU, *Second-Order Neural Nets for Constrained Optimization*, IEEE Trans. Neural Networks, Vol. 3, No. 6, 1992, pp. 1021-1024.

[272] G. ZHOU, AND J. SI, *Advanced Neural-Network Training Algorithm with Reduced Complexity Based on Jacobian Deficiency*, IEEE Trans. Neural Networks, Vol. 9, No. 3, 1998, pp. 448-453.

[273] *Verification and Validation Data for Computational Unsteady Aerodynamics*, North Atlantic Treaty Organization Research and Technology Organization, NATO-RTO-TR26, October 2000.